M4K1553PxII
&
EXC-1553PCMCIA/EPII
Galahad Software Tools
Programmer's Reference

EXCALIBUR SYSTEMS

311 Meacham Ave ♦ Elmont ♦ NY 11003 ♦ tel. (516) 327-0000 ♦ fax (516) 327-4645
e:mail: excalibur@mil-1553.com website: www.mil-1553.com

Table of Contents

1 Introduction

Getting Started 1-1
- The Excalibur Systems Website 1-2
- Installation 1-2
- The Excalibur Configuration Utility Program 1-2

An Overview of the Data Communications Bus 1-3

The MIL-STD-1760 Options 1-4
- Checksums 1-4
- Header Word 1-4

***Galahad Software Tools* Functions 1-5**
- Software Tools for the Exc-4000 Carrier Board 1-5
- Compiler Options 1-6
- Conventions Used in *Galahad Software Tools* 1-6

2 General Functions

- Enable_1553A_Support_Px 2-2
- External_Loopback_Px 2-3
- Get_Board_Status_Px 2-4
- Get_Card_Type_Px 2-4
- Get_Error_String_Px 2-5
- Get_Header_Exists_Px 2-5
- Get_Header_Value_Px 2-6
- Get_Id_Px 2-6
- Get_Mode_Px 2-7
- Get_Rev_Level_Px 2-7
- Get_Time_Tag_Px 2-8
- Init_Module_Px for PCI boards and PCMCIA cards 2-8
- Init_Module_Px for VME and VXI boards 2-10
- Internal_Loopback_Px 2-11
- Print_Error_Px 2-12
- Read_Start_Reg_Px 2-12
- Release_Module_Px 2-13
- Reset_Time_Tag_Px 2-13
- Restart_Px 2-13
- Set_Header_Exists_Px 2-14
- Set_Header_Value_Px 2-15
- Set_Interrupt_Px 2-16
- Set_Mode_Px 2-17
- Set_Timetag_Res_Px 2-18
- Stop_Px 2-18

Using Interrupts in Windows 2-19
- Get_Interrupt_Count_Px 2-20
- InitializeInterrupt_Px 2-21
- Wait_For_Interrupt_Px 2-22
- Wait_For_Multiple_Interrupts_Px 2-24

Using Interrupts Under VISA for VME Boards 2-25

3 Remote Terminal Functions

Assign_DB_Datablk_Px 3-4
Assign_RT_Data_Px 3-5
Clear_RT_Sync_Entry_Px 3-5
Get_Blknum_Px 3-6
Get_Checksum_Blocks_Px 3-6
Get_Next_Message_RTM_Px 3-7
Get_Next_RT_Message_Px 3-9
Get_RT_Message_Px 3-11
Get_RT_Sync_Entry_Px 3-12
Load_Datablk_Px 3-13
Read_Datablk_Px 3-14
Read_RT_Status_Px 3-14
Read_RTid_Px 3-15
Reset_RT_Interrupt_Px 3-15
RT_Id_Px 3-16
Run_RT_Px 3-16
Set_1553Status_Px 3-17
Set_Bit_Px 3-17
Set_Bit_Cnt_Px 3-18
Set_Broad_Interrupt_Px 3-18
Set_Checksum_Blocks_Px 3-19
Set_Invalid_Data_Res_Px 3-19
Set_Mode_Addr_Px 3-20
Set_RT_Active_Px 3-20
Set_RT_Active_Bus_Px 3-21
Set_RT_Broadcast_Px 3-21
Set_RT_Errors_Px 3-22
Set_RT_Interrupt_Px 3-23
Set_RT_Nonactive_Px 3-23
Set_RT_Resp_Time_Px 3-24
Set_RTid_Interrupt_Px 3-24
Set_RTid_Status_Px 3-25
Set_Status_Px 3-26
Set_Var_Amp_Px 3-26
Set_Vector_Px 3-27
Set_Wd_Cnt_Err_Px 3-27

4 Bus Monitor Functions

All Submodes 4-2
Clear_Msg_Blks_Px 4-2
Get_MON_Status_Px 4-3
Run_MON_Px 4-3
Set_Broad_Ctl_Px 4-4
Set_Mode_Addr_Px 4-4
Set_MON_Concurrent_Px 4-5
Set_Mon_Response_Time_Px 4-6

Look Up Submode 4-7
Assign_Blk_Px 4-7
Enable_Lkup_Int_Px 4-7
Get_Last_Blknum_Px 4-8
Get_Message_Px 4-8

Sequential Submode 4-10
Get_Counter_Px 4-10
Get_Next_Message_Px 4-10
Set_Cnt_Trig_Px 4-12
Set_Trigger_Mode_Px 4-13
Set_Trigger1_Px 4-14
Set_Trigger2_Px 4-15

5 BC/Concurrent-RT Functions

BC/Concurrent-RT Simulation 5-2
Command Word Calculation 5-2
Servicing the Service Request (SRQ) Bit 5-2
Functions by Category 5-3

BC/Concurrent-RT Functions 5-5

Alter_Cmd_Px 5-5
Alter_IMG_Px 5-6
Alter_Message_Px 5-7
Alter_MsgSendTime_Px 5-8
BC_Check_Alter_Msgentry_Px 5-9
Clear_Card_Px 5-10
Clear_Frame_Px 5-10
Clear_Timetag_Sync_Px 5-11
Command_Word_Px 5-12
Create_Frame_Px 5-13
Create_1553_Message_Px 5-14
Enable_Checksum_Px 5-15
Enable_Checksum_Error_Px 5-16
Enable_SRQ_Support_Px 5-16
Get_BC_Msgentry_Px 5-17
Get_BC_Status_Px 5-19
Get_Minor_Frame_Time_Px 5-19
Get_Msgentry_Status_Px 5-20
Get_Next_Message_BCM_Px 5-22
Insert_Msg_Err_Px 5-24
Re_Create_Message_Px 5-25
Read_Message_Px 5-26
Read_SRQ_Message_Px 5-27
Reset_BC_Status_Px 5-27
Run_BC_Px 5-28
Select_Async_Frame_Px 5-28
Send_Async_Frame_Px 5-29
Send_Timetag_Sync_Px 5-29
Set_BC_Resp_Px 5-30
Set_Bit_Cnt_Px 5-30
Set_Bus_Px 5-31
Set_Continue_Px 5-31
Set_Error_Location_Px 5-32
Set_Frame_Time_Px 5-32
Set_Halt_Px 5-33
Set_Interrupt_On_Msg_Px 5-33
Set_Jump_Px 5-34
Set_Minor_Frame_Time_Px 5-34
Set_Replay_Px 5-35
Set_Restore_Px 5-36
Set_Retry_Px 5-37
Set_RT_Active_Px 5-38
Set_RT_Nonactive_Px 5-38
Set_RT_Resp_Time_Px 5-39
Set_Skip_Px 5-39
Set_Stop_On_Error_Px 5-40
Set_Sync_Pattern_Px 5-40
Set_Var_Amp_Px 5-41
Set_Word_Cnt_Px 5-41
Set_Zero_Cross_Px 5-42
Start_Frame_Px 5-43

Appendices

Appendix A MIL-STD-1553 Formats A-1
Word Formats A-1
Message Formats A-2

Appendix B *Galahad Software Tools* Installation Instructions B-1
EXC-4000PCI Boards B-1
Windows 9x /ME Procedures B-1
Windows NT4 Procedures B-3
Windows 2000/XP Procedures B-4
EXC-4000VME Boards B-6
Installation Instructions for PCI-MXI-2 Systems B-6
PCMCIA/EPII Cards B-7
Windows 9*x*/ME Procedures B-7
Windows NT4 Procedures B-9
Windows 2000/XP Procedures B-10

Appendix C Multiple Board Support C-1
EXC-4000PCI and EXC-4000cPCI Boards C-1
EXC-1553PCMCIA/EPII Cards C-1
EXC-4000VME and EXC-4000VXI Boards C-2

Appendix D Application of External Loopback Test D-1

Appendix E Source Code References E-1
***Galahad Software Tools* Library E-2**
Code Index E-5
Error Messages E-9

Function Index . . . . . Index - 1

1 Introduction

Chapter 1 provides an overview of the Excalibur *M4K1553Px* module, (or P*x* for short) and the *EXC-1553PCMCIA/EPII* card.

Note: The P*x* module can be mounted on the following Excalibur carrier boards:

- EXC-4000PCI
- EXC-4000cPCI
- EXC-4000VME
- EXC-4000VXI

The topics covered are:

Getting Started 1-1
The Excalibur Systems Website 1-2
Installation 1-2
The Excalibur Configuration Utility Program 1-2
An Overview of the Data Communications Bus 1-3
The MIL-STD-1760 Options 1-4
Checksums 1-4
Header Word 1-4
***Galahad Software Tools* Functions 1-5**
Software Tools for the Exc-4000 Carrier Board 1-5
Compiler Options 1-6
Conventions Used in *Galahad Software Tools* 1-6

Getting Started

Welcome to *M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools*, your tool for creating custom application programs for the *M4K1553Px* module and the *EXC-1553PCMCIA/EPII* card.

Before starting to write applications:

1. On the *Excalibur Installation CD*, included in the package you received with the hardware:
   - Locate the appropriate *M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools* for your hardware;
   - Download a copy of *M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools Programmer's Reference;*
   - Download a copy of your hardware's *User's Manual*.
2. Install the hardware. For hardware installation instructions, refer to the *User's Manual*.
3. Fill out the registration card and return it to your Excalibur representative.

Note: If anything is missing or damaged, contact your Excalibur representative.

The Excalibur Systems Website

More copies of the *M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools Programmer's Reference* can be downloaded from our website: www.mil-1553.com. In addition, information about our full range of products can be found at this site.

The software provided with Excalibur hardware is for Windows operating systems. The latest versions of the software can be down loaded from our website.

For further assistance, write to excalibur@mil-1553.com.

Installation

For **PCI** and **cPCI** carrier boards, see **Appendix B-1: EXC-4000PCI Boards,** on page B-1 for hardware and software installation instructions.

For **VME** or **VXI** carrier boards, see **Appendix B-2: EXC-4000VME Boards,** on page B-6 for hardware and software installation instructions.

For PCMCIA cards, see **Appendix B-3: PCMCIA/EPII Cards,** on page B-7 for hardware and software installation instructions.

The Excalibur Configuration Utility Program

This is a configuration setup program for PCI, cPCI and PCMCIA hardware. Use the utility program to specify which hardware in a machine is being used. *M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools* supports the use of up to four PCI or cPCI boards simultaneously and two *PCMCIA/EPII* cards. Only if more than one board or card is used, is it necessary to run the utility program.

The configuration program allows the user to set or change settings on the hardware itself. Excalibur hardware will work only after device values are properly entered and saved in the **`ExcConfig`** utility. Instructions for entering values in the utility program are in:

- **Appendix C-1: EXC-4000PCI and EXC-4000cPCI Boards** or **Appendix C-2: EXC-1553PCMCIA/EPII Cards,** on page C-1 and
- In the **`readme.pdf`** file, on the *Excalibur Installation CD*, you received with your Excalibur hardware.

To configure multiple VME and VXI boards, see **Appendix C-3: EXC-4000VME and EXC-4000VXI Boards,** on page C-2.

An Overview of the Data Communications Bus

M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools enables the user to create custom diagnostic programs for the Excalibur P*x* module. To use *Galahad Software Tools*, some familiarity with the functioning of the 1553 data communications bus and an understanding of the hardware is required. This overview is an introduction to the 1553 bus. It will assist in understanding the rationale of the software functions for the hardware.

Military Standard 1553 (MIL-STD-1553) has been a mainstay of military avionics communications since its introduction in 1975. This specification defines a data communications bus capable of supporting up to 32 devices, called Remote Terminals, and coordinated by a device called a Bus Controller. The **Data communications bus** is physically composed of two wires used to transmit and receive a differential signal between the various devices. For backup purposes, a second pair of wires is often used and is called the secondary bus. The two buses are also referred to as "Bus A" and "Bus B."

Each **Remote Terminal** can transfer data to or receive data from the Bus Controller or another Remote Terminal in response to a command from the Bus Controller. Remote Terminals for their part never initiate communications but merely react to the Bus Controller.

The **Bus Controller** directs the flow of data on the data bus. Since all transmission takes place over a single pair of wires, some mechanism must be included to ensure that only a single device attempts to transmit at any given time. MIL-STD-1553 deals with this issue by dictating that the Bus Controller must initiate all communication. The Bus Controller determines the sequence of messages, the size of each message, and the timing between messages. The Bus Controller also determines which of the Remote Terminals will be transmitting or receiving data.

The **Bus Monitor** is a passive device which records 1553 bus traffic. A monitor may collect all the data from the bus or may collect selected data.

Data is transmitted in the form of **messages**. Messages consist of a Command Word containing routing information transmitted by the Bus Controller, between 1 and 32 data words containing the information to be communicated, and a Status Word transmitted by a Remote Terminal acknowledging receipt of a command. A number of message types are defined in MIL-STD-1553. The most common message types involve data transfer either from the Bus Controller (BC) to a single Remote Terminal (RT) or from an RT to the BC. Less frequently, data may be broadcast from the Bus Controller to all RTs or from one RT directly to a second RT. In all cases, the type of message is encoded in the Command Word that is transmitted by the BC.

The MIL-STD-1760 Options

MIL-STD-1760 implements an enhanced MIL-STD-1553 digital interface for the transfer of digital messages to the remote terminal. The enhancements include:

Use of Checksums	The use of checksums for the standard message, stored in place of the last data word of the message.
Support for Header Word	Support for the Header Word as the first word in a message, including the predefined Header Words.

Checksums

Checksum is mandated on critical control messages and provisional on the remainder of the messages. Implementing this level of error detection ensures a higher degree of error free data integrity requirements than that which odd parity provides.

The 1760 option implements checksum error detection capabilities. Checksums are computed as each data word is sent or received. If the checksum flag is set on an outgoing message, the checksum will be sent in place of the last data word. On an incoming message, the computed checksum is checked against the last data word received. If it does not match, the Checksum Error bit is set in the Message Status word.

The 1760 option has the additional feature of checksum error injection in BC/ Concurrent RT mode. The user can set the checksum to intentionally send an error, giving the additional capability to test for checksum errors on the receiving RTs.

Header Word

In the MIL-STD-1760 specification, the first data word of a message may be a Header Word, which is used for message identification. The header word is associated with a specific RT subaddress.

To indicate that a specific subaddress will require a Header Word, set the corresponding entry in the 1760 Header Active table to 1. Then set the corresponding entry in the 1760 Header Value table to the value you expect to receive in the first data word of the message.

The MIL-STD-1760 specification provides predefined header values. These are preset on Excalibur hardware according to the operating mode: Remote Terminal, BC/Concurrent-RT or Bus Monitor.

Galahad Software Tools Functions

M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools is a set of C language functions designed to aid users of the *Galahad Software Tools* to write test programs. These functions provide access to all of the module's functions in a structured and straightforward programming environment.

Galahad Software Tools is available for Windows 9*x*, Windows NT and Windows 2000. The Windows drivers were written and tested using Borland C++, and Microsoft Visual C++.

Software Tools for the EXC-4000 Carrier Board

There are numerous modules available for use on the EXC-4000 board. To date, Excalibur provides the following M4K modules:1553P*x*, 429/RT*x*, 1553MCH, Discrete, Serial, 708, CAN and H009.

Various combinations of these modules may be present on the carrier board at one time. One application may access any of the following configurations:

- one module
- multiple modules of the same type located on one board
- multiple modules of the same type located on separate boards
- multiple modules of different types located on one or separate boards

A general-purpose dll **`EXC4000.DLL`** (and its Microsoft-compiled version **`EXC4000MS.DLL`**) accompanies the *M4K[module] Software Tools* for each of the EXC-4000 modules. The function,Get_4000Module_Type may be called for each board and module number, to check which, if any, modules are currently available at that location. (This function is also called automatically from Init_Module*_[handle]* to ascertain that the type of module matches the module *Software Tools* package being used). These DLLs are installed automatically in the Windows/System folder when the *Software Tools* is installed for each of the modules.

The Time Tag source is set at the carrier board level, or PCMCIA card level. See the *User's Manual* for your hardware.

Note: The **`EXC4000.DLL`** is not relevant for the *EXC-1553PCMCIA/EPII* but must be present on the computer to provide initial startup services.

Compiler Options

Programmers *must* use one of the following calling options, depending on the compiler used:

- The Borland DLL is compiled using **_stdcall** options
- The Microsoft DLL is compiled using **_cdecl** options

The driver functions in *Galahad Software Tools* are supplied both in source form and linked as a DLL. When writing application-programs, keep in mind that the module is a physical resource, and therefore you cannot run multiple copies of the program simultaneously, accessing the same module.

Each function is presented with its formal definition, including data types of all input and output variables. A brief description of the purpose of the function is provided along with the legal values for inputs where applicable.

Functions are written as 'C' functions, i.e., they return values. A negative value signifies an error. Full error messages may be printed using the Get_Error_String_Px function. (See **Appendix E-3: Error Messages,** on page E-9.)

In Windows all user-defined programs must include the file **`proto_px.h`.** This file includes all the necessary header files and DLL functions to operate *M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools* for P*x* modules and P*CMCIA/EPII* cards.

Conventions Used in *Galahad Software Tools*

To help differentiate between different kinds of information, the following text styles are used in *Galahad Software Tools*:

Functions look like this.

`Variables` look like this: `unsigned short int = usint`

Parameters look like this.

`File names` look like this.

FLAGS look like this

2 General Functions

Chapter 2 contains the software functions which are not mode specific and can be used in at least two modes.

The input values included in each function are given, in Hex format, by each flag within the function description.

Any reference to a module, also applies to the *EXC-1553PCMCIA/EPII* card.

The following functions are described in this chapter:

Enable_1553A_Support_Px
External_Loopback_Px
Get_Board_Status_Px
Get_Card_Type_Px
Get_Error_String_Px
Get_Header_Exists_Px
Get_Header_Value_Px
Get_Id_Px
Get_Mode_Px
Get_Rev_Level_Px
Get_Time_Tag_Px
Init_Module_Px for PCI boards and PCMCIA cards
Init_Module_Px for VME and VXI boards
Internal_Loopback_Px
Print_Error_Px
Read_Start_Reg_Px
Release_Module_Px
Reset_Time_Tag_Px
Restart_Px
Set_Header_Exists_Px
Set_Header_Value_Px
Set_Interrupt_Px
Set_Mode_Px
Set_Header_Exists_Px
Set_Timetag_Res_Px
Set_Header_Exists_Px
Stop_Px

To handle interrupts in Windows see **Using Interrupts in Windows** on page 2-19. The functions described are:

Get_Interrupt_Count_Px
InitializeInterrupt_Px
Wait_For_Interrupt_Px
Wait_For_Multiple_Interrupts_Px

Enable_1553A_Support_Px

Description Enable_1553A_Support_Px enables MIL-STD-1553A support.

BC mode No Mode Codes with data

RT mode The Broadcast bit is not set in the 1553 Status word.

Syntax Enable_1553A_Support_Px (`int` handle, `int` enableflag)

Input Parameters

handle	The handle designated by Init_Module_Px	
enableflag	ENABLE	Enable MIL-STD-1553A support [0001 H]
	DISABLE	Disable MIL-STD-1553A support [0000 H]

Output Parameters none

Return Values

ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px
einval	If an invalid value was used as an input
emode	If current mode is not BC or RT mode
0	If successful

External_Loopback_Px

Description External_Loopback_Px is used to check the 1553 transceivers, transformers and associated bus cables.

Note External_Loopback_Px requires a loopback cable to connect Bus A to Bus B.
See **Appendix D Application of External Loopback Test** on page D-1

Syntax External_Loopback_Px (`int` handle, `struct e_loopback` *elbvals)

Input parameters

handle	The handle designated by Init_Module_Px

Output Parameters elbvals

struct E_LOOPBACK
{

	Address in Dual-Port RAM	**Status Value**
usint frame_val;	0	X (not for user)
usint frame_status;	2	8000H passed, 8001H failed
usint cmd_send[8];	4	cmd_send[0]: 5555H
	6	cmd_send[1]: 8000H passed, else failed
	8	cmd_send[2]: 1234H
	A	cmd_send[3]: 8000H passed, else failed
	C	cmd_send[4]: 5555H
	E	cmd_send[5]: 8000H passed, else failed
	10	cmd_send[6]: 1234H
	12	cmd_send[7]: 8000H passed, else failed
usint ttag_val_lo	14	30D4H ± 2
usint ttag_val_hi	16	0
usint ttag_status;	18	8000H passed, 8001H failed
} *E_loopback;		

Return values

elbfailure	If External Loopback test failed
ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px.
0	If successful

Get_Board_Status_Px

Description Get_Board_Status_Px indicates the status of the specified module.

Syntax Get_Board_Status_Px (`int` handle, `int` module)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If an invalid value was used as an input

0 or more of the following flags:

BOARD_READY	Module has completed reset sequence [0001 H]
RAM_OK	Module has passed internal memory self-test [0002 H]
TIMERS_OK	Module has passed internal timers self-test [0004 H]
SELF_TEST_OK	Module has passed full internal self-test [0008 H]
BOARD_HALTED	Module is stopped [0010 H]

Note: Bit 7 is always set to 1

Get_Card_Type_Px

Description Get_Card_Type_Px indicates the module firmware type, either 1553 or 1760.

Syntax Get_Card_Type_Px (`int` handle, `usint` *cardtype)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

cardtype	1553H or 1760H

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
0	If successful

Get_Error_String_Px

Description Get_Error_String_Px accepts the error returns from other *Galahad Software Tools* functions. This function returns the string containing a corresponding error message.

See **Appendix E-3 Error Messages** on page E-9.

Syntax Get_Error_String_Px (`int` errcode, `char` *errstring)

Example

```
char ErrorStr[255];
Get_Error_String_Px (errorcode, ErrorStr);
printf("error is: %s", ErrorStr)
```

Input Parameters	errorcode	The error code returned from a *Galahad Software Tools* function.
Output Parameters	errstring	A string of characters, with the corresponding error message. In case of bad input, a string denoting that.
Return Values	0	Always

Get_Header_Exists_Px

Description Get_Header_Exists_Px checks to see if the Subaddress (SA) has an assigned Header Word Value.

Note: This function is applicable *only* to modules that support the 1760 options

Syntax Get_Header_Exists_Px (`int` handle, `int` sa, `int` *enable)

Input Parameters	handle	The handle designated by Init_Module_Px
	sa	The Subaddress to be checked
Output Parameters	enable:	Legal flags are:
		1 Header Word assigned
		0 Header Word *not* assigned
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	rterror	Bad Subaddress number
	0	If successful

Get_Header_Value_Px

Description Get_Header_Value_Px gets the Header Word assigned to this Subaddress.

Note: This function is applicable *only* to modules that support the 1760 options.

Syntax Get_Header_Value_Px (`int` handle, int `sa`, usint *header_value)

Input Parameters

handle	The handle designated by Init_Module_Px
sa	The Subaddress to be checked

Output Parameters

header_value	The assigned Header Word

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
rterror	Bad Subaddress number
0	If successful

Get_Id_Px

Description Get_Id_Px returns the module ID of the specified module.

Syntax Get_Id_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
Anything other than module ID	Failure
Module ID: E [0045 H]	If successful

Get_Mode_Px

Description Get_Mode_Px indicates the current mode of the specified module reflecting the last call to Set_Mode_Px.

Syntax Get_Mode_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If an invalid value was used as an input

One of the following flags:

RT_MODE	Module set up as a Remote Terminal [0002 H]
BC_RT_MODE	Module set up as BC/Concurrent RT [0004 H]
MON_BLOCK	Module set up as a Monitor in Sequential Block submode [0008 H]
MON_LOOKUP	Module set up as Monitor in Table Lookup submode [0020 H]

Get_Rev_Level_Px

Description Get_Rev_Level_Px indicates the current revision level of the firmware for the specified module. Each Hex nibble (i.e. 4 bits) reflects a level, so the 0x11 should be interpreted as revision 1.1.

Syntax Get_Rev_Level_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
firmware revision level	If successful

Get_Time_Tag_Px

Description	Get_Time_Tag_Px returns the running Time tag of the module.	
Syntax	Get_Time_Tag_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	32 bit Time-tag	If successful

Init_Module_Px for PCI boards and PCMCIA cards

Description

Init_Module_Px is the first function the user must call for each P*x* module, on each board, or *PCMCIA/EPII* card that is accessed in the application program.

EXC-4000PCI and cPCI

Init_Module_Px enables the user to access up to four modules on a single board, or any combination of up to 16 modules on four separate boards.

The function may be called with the SIMULATE argument. If the SIMULATE argument is used, a portion of the memory equal to the size of the board's dual-port RAM is set aside. This area is then initialized with an id and version number for use in testing programs when no module is available.

Multiple modules may be simulated. It is possible to have real or simulated modules in one application. Up to 17 real or simulated modules may be initialized. More than one module may be simulated simultaneously.

EXC-1553PCMCIA/EPII

Up to two cards may be used.

The function may be called with the SIMULATE argument. If the SIMULATE argument is used, a portion of the memory equal to the size of the card's dual-port RAM is set aside. This area is then initialized with an id and version number for use in testing programs when no card is available.

Two cards may be simulated simultaneously. It is possible to have real or simulated cards in one application.

Note: Before exiting a program, call Release_Module_Px for each module initialized with Init_Module_Px.

Syntax

Init_Module_Px (`usint` device_num, `usint` module_num)

Init_Module_Px (cont.)

Input parameters		**EXC-4000PCI and cPCI**
	device_num	The device number is the index of the entry value set in `ExcConfig`: 0 - 3 *or* SIMULATE [FFFF H]
	module_num	The module number of the P*x* module on the board specified by device_num: 0 – 3 **EXC-1553PCMCIA/EPII** EXC_PCMCIA_MODULE [0064 H]
Note:	If only one board is used, the define value EXC_4000PCI can be used instead of a device number. If more than one board is used the programmer must run the `ExcConfig` utility to set the device number.	
Output parameters	none	
Return Values	sim_no_mem	If init failed in SIMULATE mode due to Malloc call failure
	eopenkernel	If there was an error opening a device
	ekernelcantmap	If a pointer to memory cannot be obtained
	emodnum	Invalid module number specified
	enomodule	If no EXC-4000 module present at specified location
	ewrngmodule	If module specified on EXC-4000PCI board is not P*x* module
	enoid	If init could not find a module with the given device number
	etimeoutreset	If timed out waiting for reset
	ealocresources	If there was an error allocating resources
	handle	If successful, the handle to the specified module on the board. This handle is used as the first parameter in all module functions. A valid handle is a positive number ranging from 0 – 16.

Init_Module_Px for VME and VXI boards

Description

Init_Module_Px for VME is the first function the user must call for a VME or VXI board. The function may be called with the SIMULATE argument.

Init_Module_Px must be called once per module (in multi-module) applications. Up to 8 modules can be used or simulated per board. The number of VME or VXI boards which can be used or simulated simultaneously, depends on the system used.

Before exiting a program, call Release_Module_Px for each module for which Init_Module_Px was called.

Syntax

Init_Module_Px (`usint` device_num, `usint` module_num)

Input Parameters

device_num — A device number (0 – 255) as set with the DIP switches.
or
SIMULATE [FFFF H]

See **Appendix B-2 EXC-4000VME Boards** on page B-6.

module_num — The module number of the P*x* module on the board specified by device_num: 0 – 7

Output Parameters

none

Return Values

eboardnotfound	If device number is not valid
sim_no_mem	If init failed in SIMULATE mode due to Malloc call failure
enoid	If init could not find a board with the given device number as set by the DIP switches.
etimeoutreset	If timed out waiting for reset
eallocresources	If there was an error allocating resources
eopendefaultrm	If error in VISA function viOpenDefaultRM
eviopen	If error in VISA function viOpen
evimapaddress	If error in VISA function viMapAddress
0	If successful

Internal_Loopback_Px

Description Internal_Loopback_Px is used to check the 1553 front-end logic, excluding transceivers and coupling transformers.

Syntax Internal_Loopback_Px (`int` handle, `struct i_loopback` *ilbvals)

Input parameters

handle	The handle designated by Init_Module_Px

Output parameters ilbvals

struct I_LOOPBACK

{

	Address in Dual-Port Ram	Status Value
usint frame_val;	0	X (not for user)
usint frame_status;	2	8000H passed, 8001H failed
usint resp_status;	4	8000H passed, 8001H failed
usint early_val;	6	6 LSB must be 15H
usint receive_data1;	8	5555H
usint status_1;	A	8000H passed, else failed
usint receive_data2;	C	AAAAH
usint status_2;	E	8000H passed, else failed
usint mc_status;	10	8000H passed, else failed
usint ttag_val_lo;	12	30D4H ± 2
usint ttag_val_hi;	14	0
usint ttag_status;	16	8000H passed, 8001H failed
usint prl;	18	8 LSB contain the CPU version

} *I_loopback;

Return values

ilbfailure	If Internal Loopback test failed.
ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px.
0	If successful

Print_Error_Px

Description Print_Error_Px has been superseded by Get_Print_Error_Px. The function has been retained for backward compatibility.

Print_Error_Px accepts the error code returned from other *Galahad Software Tools* functions. This function returns a char pointer to a string containing a corresponding error message. (See **Appendix E-3 Error Messages** on page E-9 for a complete list of error messages.)

Syntax char * Print_Error (`int` errorcode)

Input Parameters	errorcode	The error code returned from a *Galahad Software Tools* call
Output Parameters	none	
Return Values	a pointer	Pointer to a message string that contains a corresponding error message. In case of bad input this function points to a string denoting "No such message".

Read_Start_Reg_Px

Description Read_Start_Reg_Px returns the value in the Start register, indicating whether or not the module is currently operating.

Syntax Read_Start_Reg_Px (`int` handle, `usint` *startval)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	startval	Value in the start register
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Release_Module_Px

Description	Release_Module_Px must be called for each module initialized with Init_Module_Px, before exiting a program. Following a call to this function, the user must call Init_Module_Px before selecting another module or release memory allocated for simulated modules.	
Syntax	Release_Module_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Reset_Time_Tag_Px

Description	Reset_Time_Tag_Px resets the module's Time tag register.	
Syntax	Reset_Time_Tag_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Restart_Px

Description	Restart_Px continues processing following Set_Stop_on_Error_Px.	
Syntax	Restart_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Set_Header_Exists_Px

Description

Set_Header_Exists_Px indicates that messages associated with this Subaddress are expected to have the specified Header Word.

Note: This function is applicable *only* to modules that support the 1760 options.

Syntax

Set_Header_Value_Px (`int` handle, `int` sa, `usint` enable)

Input Parameters

handle	The handle designated by Init_Module_Px		
sa	The Subaddress assigned a Header Word		
enable	Valid values depend of the current mode:		
BC/Concurrent-RT mode	HEADER_ENABLE	Header Word expected on Transmit messages [0001 H]
	HEADER_DISABLE	No Header Word expected [0000 H]
RT mode	HEADER_ENABLE	Header Word expected on Receive messages [0001 H]
	HEADER_DISABLE	No Header Word expected [0000 H]
Monitor mode	HEADER_ENABLE_RCV	Header Word expected on Receive messages [0003 H]
	HEADER_ENABLE_XMT	Header Word expected on Transmit messages [0002 H]
	HEADER_ENABLE	Header Word expected on both Transmit and Receive messages [0001 H]
	HEADER_DISABLE	No Header Word expected [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
rterror	If a bad Subaddress number is selected
0	If successful

Set_Header_Value_Px

Descriptionf	Set_Header_Value_Px sets the Header Word assigned to this Subaddress.	
Note	This function is applicable *only* to modules that support the 1760 options.	
Syntax	Set_Header_Value_Px (`int` handle, `int` sa, `usint` header_value)	
Input Parameters	handle	The handle designated by Init_Module_Px
	sa	The Subaddress associated with this Header Word
	header_value	The assigned Header Value
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	rterror	If a bad Subaddress number is selected
	0	If successful

Set_Interrupt_Px

Description

Set_Interrupt_Px describes the conditions under which the module is to generate an interrupt or disables interrupts.

Note: In RT mode, interrupts can also be generated using Set_RT_Interrupt_Px. See Set_RT_Interrupt_Px, page-3-23.

Syntax

Set_Interrupt_Px (`int` handle, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px	
flag	Conditions must be appropriate to the current mode. Multiple conditions (flags) maybe ORed together	
	0	Do not set interrupts of any kind
BC/Concurrent-RT mode	MSG_CMPLT	Message sent [0002 H]
	END OF FRAME	Complete frame of messages sent [0004 H]
	MSG_ERR	Error occurred [0008 H]
	SRQ_MSG	Interrupt when SRQ messages sent [0020 H]
	END_MINOR_FRAME	Minor Frame completed [0010 H]
RT mode	MSG_CMPLT	Message sent [0002 H]
	BEGIN_MSG	Start of message sent [0001 H]
Sequential Monitor mode	TRIG_RCVD	Received message, matched trigger [0001 H]
	MSG_IN_PROGRESS	Message in progress [0002 H]
	CNT_TRIG_MATCH	Set_Cnt_Trig_Px block number message received [0004 H]
Look-up Table Monitor	MSG_IN_PROGRESS	Message in progress [0002 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
eintr	If attempted to set an undefined interrupt
noirqset	If no interrupt allocated
emode	If the module mode is not recognized
0	If successful

Set_Mode_Px

Description

Set_Mode_Px sets the mode that the relevant module on the board is to operate. It performs a reset, clearing all the memory on the module and initializes the module to its default values.

This function must be called prior to any mode specific function.

Note: Set_Mode_Px may be called just for reset purposes; if you wish to reset *only* the function variables in BC mode, Clear_Card_Px or Clear_Frame_Px are faster.

Syntax

Set_Mode_Px (`int` handle, `int` mode)

Input Parameters

handle	The handle designated by Init_Module_Px
mode	
BC_RT_MODE	Set up module as a BC/Concurrent-RT [0004 H]
RT_MODE	Set up module as Remote Terminal [0002 H]
MON_BLOCK	Set up module as Monitor in Sequential Block submode [0008 H]
MON_LOOKUP	Set up module as Monitor in Table Lookup submode [0020 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If an invalid parameter was used as an input
bad_mem	If an invalid pointer was specified
etimeoutreset	If timed out waiting for reset
0	If successful

Set_Timetag_Res_Px

Description Set_Timetag_Res_Px sets the resolution of the time tag used in the RT and Monitor Mode. The minimum Time tag resolution is 4 μsec. Resolution is rounded up the nearest multiple of 4.

Note: Set Set_Timetag_Res_Px before starting RT or Monitor mode. The new resolution takes effect the next time the RT or Monitor is started.

Syntax Set_Timetag_Res_Px (`int` handle, `usint` resolution)

Input Parameters	handle	The handle designated by Init_Module_Px
	resolution	Valid values: 4 – 1024, multiples of 4 only
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	einval	If an invalid parameter was used as an input
	0	If successful

Stop_Px

Description Stop_Px stops the current operation of the module in all modes.

Syntax Stop_Px (`int` handle)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Using Interrupts in Windows

When writing a Windows program that processes interrupts, a separate thread is generally created to handle the interrupt processing. This thread calls Wait_For_Interrupt_Px, page-2-22, in order to wait for the next interrupt. When the function returns, the interrupt is processed as needed. This method is demonstrated in the test programs **`demo_int.c`** and **`demo_intms.c`** included with the *Galahad Software Tools.*

Note: There is no need to reset the physical interrupt line in the interrupt thread; this is handled internally.

In cases of very high interrupt frequency, several interrupts may occur before the interrupt thread resumes execution. The Get_Interrupt_Count_Px function may be used to determine if multiple interrupts have occurred. Conversely, it is possible that the Wait_For_Interrupt_Px function will indicate an interrupt that has already been processed by the thread. (This will occur in the case where a subsequent interrupt occurs in between the return of the Wait_For_Interrupt_Px function and the call to Get_Interrupt_Count_Px.) Once again, the Get_Interrupt_Count_Px function may be used to determine if the interrupt has already been processed.

The following functions are described below:

Get_Interrupt_Count_Px
InitializeInterrupt_Px
Wait_For_Interrupt_Px
Wait_For_Multiple_Interrupts_Px

Get_Interrupt_Count_Px

Description

Get_Interrupt_Count_Px returns the total interrupt count for the specified module from the time the module was initialized with Init_Module_Px.

Syntax

Get_Interrupt_Count_Px (`int` handle, `unsigned long` *pdwInterruptCount)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

pdwInterruptCount	Pointer to an unsigned long which receives the interrrupt count

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
egetintcount	If there was a kernel error
ekernelinitmodule	If error initializing kernel related data
ekernelbadparameter	If input parameter is invalid
ekernelbadpointer	If output parameter buffer is invalid
ekerneldevicenotopen	If specified device has not been opened
0	If successful

InitializeInterrupt_Px

Description

InitializeInterrupt_Px instructs the Excalibur kernel driver to keep track of interrupts which occur on the Excalibur module. Once this function has been called, the kernel driver will make note of any interrupts which occur on the module, even if the application is not currently waiting for interrupts (via the Wait_For_Interrupt_Px function). When the application later calls Wait_For_Interrupt_Px, the function will return immediately, indicating an interrupt has occurred in the meantime.

Note: 1. This function need only be called once during a given session with a module. Subsequently, interrupts will continue to be tracked for the module until Release_Module_Px is called to close the session.

2. The use of this function is not absolutely necessary for interrupt processing. If Wait_For_Interrupt_Px is called before InitializeInterrupt_Px has been called, then the initialization will be performed automatically, at that point. The use of InitializeInterrupt_Px is necessary only if the user requires interrupt tracking prior to the first call to Wait_For_Interrupt_Px.

Syntax

InitializeInterrupt_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
egeteventhandle1	If there is an error in kernel Get_Event_Handle, first part
egeteventhandle2	If there is an error in kernel Get_Event_Handle, second part
ekernelinitmodule	If error initializing kernel related data
ekernelbadparam	If input parameter is invalid
ekerneldevicenotopen	If specified device was not opened
ekernelbadpointer	If output parameter buffer is invalid
0	If successful

Wait_For_Interrupt_Px

Description

Wait_For_Interrupt_Px waits for an interrupt on the module. It suspends control of the calling thread while waiting, and returns control to the thread upon receipt of the interrupt, or upon expiration of the time out. If timeout is set to INFINITE, then the call will return only upon receipt of the interrupt.

Syntax

Wait_For_Interrupt_Px (`int` handle, `unsigned int` timeout)

Example

Since this function suspends execution of the calling thread, it is generally called from a separate thread, to allow the main thread to continue its processing. An example of a thread routine which waits for interrupts and processes them as they come in is as follows:

```
DWORD InterruptThread(int referenceParam)
{
  while (1)
  {
    int status;
    status = Wait_For_Interrupt_Px(module_handle, INFINTE);
    if (status < 0)
    {
      // We don't check for ekerneltimeout since we passed
      // in a timeout value of INFINITE.
      // All other return values indicate error.
      // Process error…
       ExitThread(1);
    }
    // Process interrupt…
    // Get the status register to determine the
    //cause of this interrupt
    bc_stat = Get_BC_Status;
    Reset_BC_Status();

    if (bc_stat & END_OF_FRAME)
    {
    // interrupt due to end of frame
    // process…
    }
    if (bc_stat & MSG_CMPLT)
    {
    // interrupt due to message completed
    // process…
    }
    if (bc_stat & MSG_ERR)
    {
    // interrupt due to message error
    // process…
    }
    // Check total number of interrupts
    Get_Interrupt_Count_Px(module_handle, &numints);
  }
}
```

Wait_For_Interrupt_Px (cont.)

Input Parameters	handle	The handle designated by Init_Module_Px
	timeout	Timeout is specified in milliseconds *or* INFINITE [FFFF H]
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle is specified; should be value returned by Init_Module_Px
	egeteventhand1	If there is an error in kernel mGetEventHandle, first part
	egeteventhand2	If there is an error in kernel mGetEventHandle, second part
	ekernelinitmodule	If error initializing kernel related data
	ekernelbadparam	If input parameter is invalid
	ekerneldevicenotopen	If specified device was not opened
	Successful if *either*:	
	ekerneltimeout	The wait timed out without receiving an interrupt
	or	
	0	

Wait_For_Multiple_Interrupts_Px

Description

Wait_For_Multiple_Interrupts_Px waits for an interrupt on any of the specified modules. It suspends control of the calling thread while waiting, and returns control to the thread either upon receipt of the interrupt, or upon expiration of the time out. If timeout is set to INFINITE, then the call will return only upon receipt of the interrupt.

Syntax

Wait_For_Multiple_Interrupts_Px(`int` *handle_list, `int` num_modules, `unsigned int` timeout, `unsigned long` *pdwInterruptBitfield)

Input Parameters

handle_list	An array of module handles
num_modules	Number of modules in the handle_list
timeout	Timeout is specified in milliseconds *or* INFINITE [FFFF H]

Output Parameters

pdwInterruptBitfield	Pointer to an unsigned long which receives a bitfield indicating which of the modules have interrupted (note that more than one module may have interrupted simultaneously). The modules are distributed in the bitfield such that the lowest bit corresponds to the first module in the handle_list, and so on.

Return Values

egeteventhand1	If there is an error in kernel mGetEventHandleForModule, first part
egeteventhand2	If there is an error in kernel mGetEventHandleForModule, second part
ebadhandle	If invalid handle specified; should be value returned by Init_Module_RTx.
ekernelinitmodule	If error initializing kernel related data
ekernelbadparam	If input parameter is invalid
ekerneldevicenotopen	If the specified device was not opened
ekernelbadpointer	If output parameter buffer is invalid
Successful if *either*:	
ekerneltimeout	The wait timed out without receiving an interrupt
or	
0	

Using Interrupts Under VISA for VME Boards

When using interrupts under VISA an “interrupt service function” must be created into which the user can place any code that is to run in response to receiving an interrupt. The interrupt line should be reset when an interrupt is received so that the next interrupt can be recorded.

Galahad Software Tools includes test programs, such as **`demo_int.c`** and **`demo_intms.c`**, that show how to use interrupts using VISA library functions.

3 Remote Terminal Functions

Chapter 3 describes the Excalibur P*x* module and *PCMCIA/EPII* card operation in Remote Terminal (RT) mode. The RT module is used to simulate one or more Remote Terminals. The user can:

- Control which RTs will be simulated.
- Determine which data should be sent for each RT/subaddress combination.
- Store data to each RT/subaddress combination.

Any reference to a module, also applies to the *EXC-1553PCMCIA/EPII* card.

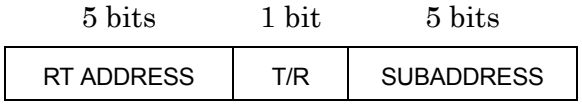
The functions described in this chapter are:

Assign_DB_Datablk_Px
Assign_RT_Data_Px
Get_Blknum_Px
Get_Checksum_Blocks_Px
Get_Next_Message_RTM_Px
Get_Next_RT_Message_Px
Get_RT_Message_Px
Load_Datablk_Px
Read_Datablk_Px
Read_RT_Status_Px
Read_RTid_Px
Reset_RT_Interrupt_Px
RT_Id_Px
Run_RT_Px
Set_1553Status_Px
Set_Bit_Px
Set_Bit_Cnt_Px
Set_Broad_Interrupt_Px
Set_Checksum_Blocks_Px
Set_Invalid_Data_Res_Px
Set_Mode_Addr_Px
Set_RT_Active_Px
Set_RT_Active_Bus_Px
Set_RT_Broadcast_Px
Set_RT_Errors_Px
Set_RT_Interrupt_Px
Set_RT_Nonactive_Px
Set_RT_Resp_Time_Px
Set_RTid_Interrupt_Px
Set_RTid_Status_Px
Set_Status_Px
Set_Var_Amp_Px
Set_Vector_Px
Set_Wd_Cnt_Err_Px
Set_Checksum_Blocks_Px

The input values included in each function are given, in Hex format, by each flag within the function description.

RT Identifier

Many of the functions in this section take an RT identifier (RTid) as an argument. The RTid is defined as an RT address, T/R bit value and RT subaddress combination. The structure of the RTid is illustrated below:



Example: RT 5, Transmit, Subaddress 6 would be represented as 00101 1 00110 or 0166 H. This value can be isolated from a Command word by shifting the Command word 5 bits to the right.

The RT_Id_Px function (on page 3-16) is provided to carry out this calculation.

For quick reference, the frequently called functions are grouped by use:

Init_Module_Px	To initialize the module
Set_Mode_Px	To place the module into RT mode
Set_RT_Active_Px	To select which RT(s) to simulate
Run_RT_Px	To start simulation
Stop_Px	To Stop the RT
Release_Module_Px	To release resources assigned to the module

To control the data being transmitted by the simulated RT(s) or read the data received by them:

Assign_RT_Data_Px	To associate a buffer with an RTid
Load_Datablk_Px	To enter data to be transmitted
Read_Datablk_Px	To read data received by an RTid
Set_1553Status_Px	To set up a value to be transmitted as the 1553 Status Word for the specified RT and to be used as a response to the Transmit Status mode commands
Set_Status_Px	To set up response to Transmit Status mode commands and to set up the 1553 Status Word

To use messages in the Message Stack:

A Command stack retains a history of the last 512 commands processed by the module. This history is used to compile statistics about bus traffic and alerts the user to incoming messages so that any necessary processing can be performed in real-time. Use of interrupts notifies the user when a particular message comes in and can save processing time. The functions to use are:

Get_Next_RT_Message_Px	To read an entry from the message stack
Set_RT_Interrupt_Px	To select which RTs generate interrupts

To invoke Error injection capabilities:

The P*x* module provides a wide array of error injection capabilities for testing at a systems level. To invoke error injection capabilities call:

Set_Bit_Cnt_Px	To alter the number of bits in transmitted data
Set_RT_Errors_Px	To select which errors to inject
Set_Wd_Cnt_Err_Px	To select how many words to send in a message

To simulate other systems:

A number of system parameters may be altered to change the character of the system.

Set_Invalid_Data_Res_Px	To select how to respond to invalid data
Set_Mode_Addr_Px	To select which subaddress represents a mode command
Set_RT_Resp_Time_Px	To select how long to wait before responding to a command

To send checksum values in modules with the 1760 options:

With the Excalibur P*x* modules with the 1760 options, the user can set which data blocks (associated with some RTids) are associated with a valid checksum value.

Set_Checksum_Blocks_Px	To set which block numbers are enabled to check for a valid checksum value

Assign_DB_Datablk_Px

Description

Assign_DB_Datablk_Px enables double buffering for the specified Receive type RTid and assigns the data blocks for the messages received by that RTid. The data for each message received by the RTid will be stored alternately in the given block and in the next block sequence. The block number specified must be an even-numbered block.

For example, selecting block number 4 will cause that the data for messages to this RTid will be stored alternately in block 4, block 5, block 4, block 5, etc.

This data can be retrieved by the user via the Read_RTid_Px function.

Syntax

Assign_DB_Datablk_Px (`int` handle, `int` rtid, `int` enable, `int` blknum)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier on page 3-2 and RT_Id_Px on page 3-16.	
enable	ENABLE	Enable double buffering for the RTid [0001 H]
	DISABLE	Disable double buffering for the RTid [0001 H]
blknum	Even numbered Data block: 0 – 199 **Note:** Data block 0, while technically a legal block, should be avoided. It is the default block for all RTs and all active RTs with no block assigned will end up here.	

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
ercvfunc	If T/R bit is set to Transmit
ebadblknum	If block number selected is not an even number
0	If successful

Assign_RT_Data_Px

Description Assign_RT_Data_Px associates an RTid with a data block.

Syntax Assign_RT_Data_Px (`int` handle, `int` rtid, `int` blknum)

Input Parameters

handle	The handle designated by Init_Module_Px
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier on page 3-2 and RT_Id_Px on page 3-16.
blknum	Data block 0 – 199 **Note:** Data block 0 represents a default for all unassigned RTid's and is not recommended for use by anyone interested in using the data. If the RT does not assign a Data block to an RTid, the default Data block is used for both receive and transmit messages.

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Clear_RT_Sync_Entry_Px

Description Clear_RT_Sync_Entry_Px clears (resets to 0) the internally stored Time tag and Data word, if any, relating to the occurrence of a Sync Mode Code message (MC-1 or MC-17) for this RT.

Syntax Clear_RT_Sync_Entry_Px (`int` handle, `int` rtnum)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	The number of the RT for which to get the Sync information. Valid values: 0 – 31

Output Parameters `none`

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode.
einval	If parameter is out of range
0	If successful

Get_Blknum_Px

Description	Get_Blknum_Px returns the data block number which was assigned to specified RTid.	
Syntax	Get_Blknum_Px (`int` handle, `int` rtid)	
Input Parameters	handle	The handle designated by Init_Module_Px
	rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier on page 3-2 and RT_Id_Px on page 3-16.
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	einval	If parameter is out of range
	block number	If successful: valid values 0 – 199

Get_Checksum_Blocks_Px

Description	Get_Checksum_Blocks_Px returns the number of data blocks for which the associated RT subaddress will receive a checksum. This function returns the value set by Set_Checksum_Blocks_Px on page 3-19. **Note:** This function is applicable *only* to modules that support the Excalibur 1760 options.	
Syntax	Get_Checksum_Blocks_Px (`int` handle)	
Input Parametersi	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	Invalid value or no value set
	1 – 200	Number of data blocks to receive a checksum. For example: A value of 5 means that data blocks 0 – 4 will receive a checksum; 5 – 199 will not.

Get_Next_Message_RTM_Px

Description

Get_Next_Message_RTM_Px relates to the **Internal Concurrent Monitor** available in the memory of the module. The **Internal Concurrent Monitor** operates automatically when the module is started in Remote Terminal mode. All bus traffic in RT mode is recorded in the **Internal Concurrent Monitor** memory. The Concurrent Monitor operates in sequential fixed block mode which means that the 1553 Message blocks are stored in sequential locations in memory, beginning from block 0.

The first call to the function Get_Next_Message_RTM_Px returns the contents of block 0. If the next block does not contain a new message, an error value will be returned.

Note: The structure returned by this function is similar to that returned by the Sequential Monitor mode function Get_Next_Message_Px. However, the bits used in the Message Status word of the **Internal Concurrent Monitor** are unique to this function – they are similar but not identical to the bits set in the Message Status word of Sequential Monitor mode and BC/Concurrent-RT mode and its Internal Concurrent Monitor.

For each mode verify the Message Status bits for the appropriate mode and function.

Syntax

Get_Next_Message_RTM_Px (`int` handle, `struct MONMSG` *msgptr, `usint *`param3)

Input Parameters

handle — The handle designated by Init_Module_Px

Output Parameters

msgptr — A pointer to a structure of type MONMSG containing information regarding the next available entry from the message stack.

```
struct MONMSG {
  usint msgstatus;        Status Word containing one or more
                          of the following flags:
    END_OF_MSG            Indicates end of message
                          [8000 H]
    BUS_A                 Indicates bus A [4000 H]
    BAD_CHECKSUM_CONCM    Checksum error [2000 H]
                          (for 1760 option only)
    ME_SET                Message Error bit in the RT
                          Status word [1000 H]
    BAD_STATUS            RT Status Word bits set (not
                          ME bit) [0800 H]
```

Get_Next_Message_RTM_Px (cont.)

	TX_TIME_OUT	In RT-to-RT message receiving RT did not detect a transmitter Status Word [0400 H]
	LATE_RESP	Response time error occurred in the message, even if no RT active on the module [0200 H]
	INVALID_MSG_RT2RT	Invalid message; perhaps RT-to-RT with two receive Command Words [0100 H]
	INVALID_WORD [0080 H]	At least one invalid 1553 Word received [0080 H]
	BAD_HEADER_RTCM	Bad Header Word [0040 H] (for 1760 option only)
	WORD_CNT_ERR	Incorrect number of words received in a message [0020 H]
	BAD_RT_ADDR	Received 1553 Status word did not contain the correct RT address [0010 H]
	BAD_SYNC	Sync of either the Command or the Data word(s) is incorrect [0008 H]
	BAD_GAP	Invalid gap received between 1553 Words [0004 H]
	RT2RT_MSG_CONCM	RT-to-RT message was received [0002 H]
	MSG_ERROR	Error occurred - defined in other flags [0001 H]
unsigned long elapsedtime;	A 32-bit-Time Tag	
unsigned int words [36];	A pointer to an array of 1553 Words in the sequence they were received over the bus.	
}		
param3	Reserved for future use	

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
enomsg	If there is no message to return
block number	If successful. Valid values: 0 – 127

Get_Next_RT_Message_Px

Description

Get_Next_RT_Message_Px reads the next available entry from the message stack; the entry immediately following the entry that was returned via a previous call to this routine. If the next block does not contain a new entry an error will be returned.

Note: If the user falls 512 entries behind, it will be impossible to know if the message returned is newer or older than the previous message returned.

Syntax

Get_Next_RT_Message_Px (`int` handle, `struct CMDENTRYRT` *cmdstruct)

Input Parameters

handle — The handle designated by Init_Module_Px

Output Parameters

A pointer to a structure of type CMDENTRY containing information regarding the next available entry from the message stack.

struct CMDENTRYRT *cmdstruct

struct CMDENTRYRT{

usint command;	1553 Command word
usint command2;	1553 transmit Command word for RT-to-RT
usint timetaghi;	16 upper bits of 32-bit Time tag of message
usint timetaglo;	16 lower bits of 32-bit Time tag of message
usint status;	status containing one or more of the following flags:

END_OF_MSG	Indicates end of message [8000 H]
BUS_A	Indicates bus A [4000 H]
RT_BAD_CHECKSUM	Bad 1760 Checksum on the RT receive side (for 1760 only) [2000 H]
TX_TIME_OUT	In RT-to-RT no response [0400 H]
INVALID_WORD	At least one invalid 1553 Word received [0080 H]
BAD_WD_CNT	Wrong number of words received [0020 H]
BROADCAST	Broadcast command received [0010 H]
BAD_SYNC	Sync of either the Command or the Data word(s) is incorrect [0008 H]
BAD_GAP	Invalid gap received between 1553 Words [0004 H]

Get_Next_RT_Message_Px (cont.)

RT2RTMSG	RT-to-RT message was received [[0002 H]
MSG_ERROR	Error occurred - defined in other flags [0001 H]

}

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
enomsg	If no unread messages are available to return
entry number	If successful: valid values are 0 – 511

Get_RT_Message_Px

Description

Get_RT_Message_Px reads the next available entry from the message stack; the entry immediately following the entry that was returned via a previous call to this routine. If the next block does not contain a new entry an error will be returned.

Note:

1. If the user falls 42 entries behind, it will be impossible to know if the message returned is newer or older than the previous message returned.
2. Call this function for modules rev A, A1 and C using firmware 1.6 or earlier.
   Call Get_Next_RT_Message_Px for module rev C using firmware 1.7 or later.

Syntax

Get_RT_Message_Px (`int` handle, `struct CMDENTRY` *cmdstruct)

Input Parameters

handle The handle designated by Init_Module_Px

Output Parameters

A pointer to a structure of type CMDENTRY containing information regarding the next available entry from the message stack.

struct CMDENTRY *cmdstruct

struct CMDENTRY{

usint command;	1553 Command word
usint command2;	1553 transmit Command word for RT-to-RT
usint timetag;	16-bit Time tag of message
usint status;	status containing the following flags:

END_OF_MSG	Indicates end of message [8000 H]
BUS_A	Indicates bus A [4000 H]
RT_BAD_CHECKSUM	Bad Checksum on the RT receive side [2000 H] (for 1760 only)
TX_TIME_OUT	In RT-to-RT no response [0400 H]
INVALID_WORD	At least one invalid 1553 Word received [0080 H]
BAD_WD_CNT	Wrong number of words received [0020 H]
BROADCAST	Broadcast command received [0010 H]
BAD_SYNC	Sync of either the Command or the Data word(s) is incorrect [0008 H]
BAD_GAP	Invalid gap received between 1553 Words [0004 H]

Get_RT_Message_Px (cont.)

RT2RTMSG	RT-to-RT message was received [0002 H]
MSG_ERROR	Error occurred - defined in other flags [0001 H]

}

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
enomsg	If no unread messages are available to return
entry number	If successful Values values: 0 – 41

Get_RT_Sync_Entry_Px

Description

Get_RT_Sync_Entry_Px extracts the internally stored Time tag and Data word, if any, of the most recent occurrence of a Sync Mode Code message (MC-1 or MC-17) for this RT.

Syntax

Get_RT_Sync_Entry_Px (`int` handle, `int` rtnum, `usint` *sync_timetag, `usint` *sync_data)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	The number of the RT for which to get the Sync information. Valid values: 0 – 31

Output Parameters

sync_timetag	The lower 16-bits of the Time tag reading when the Sync message arrived
sync_data	The Data Word of the Sync message. For Sync message with data, MC-17, only.

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode.
einval	If parameter is out of range
0	If successful

Load_Datablk_Px

Description

Load_Datablk_Px assigns data to a specified data block. This data will be used to respond to a Transmit command or a Transmit mode command.

Note: The same data block may be assigned to multiple RTs. Data block 0 is the default data block used by any Active RT for which Assign_RT_Data_Px has not been called. Since this block is the default for Receive commands as well, its data may change in an unpredictable way. Avoid using this block.

Syntax

Load_Datablk_Px (`int` handle, `int` blknum, `usint` *words)

Input Parameters

handle	The handle designated by Init_Module_Px
blknum	Number of block to assign data to. Valid values: 0 – 199

Output Parameters

words	Pointer to an array of up to 32 words of data to be placed in the block

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Read_Datablk_Px

Description Read_Datablk_Px returns the number of words requested from a particular block. The most common use of this function is to retrieve data from a block assigned to a receive RTid. To do this, set up the block and check Read_RT_Status_Px (or wait for an interrupt). When a message is received, call Get_RT_Message_Px or Get_Next_RT_Message_Px to find out which RT received the message. Then call Read_Datablk_Px to retrieve the data.

Syntax Read_Datablk_Px (`int` handle, `int` blknum, `usint` *words)

Input Parameters

handle	The handle designated by Init_Module_Px
blknum	Number of block to assign data to. Valid values: 0 – 199

Note: Data block 0, while technically a legal block, should be avoided. It is the default block for all RTs and all active RTs with no block assigned will end up here.

Output Parameters

words	Pointer to an array of up to 32 words of data to be placed in the block. Although the message may contain fewer than 32 words, 32 words will be transferred.

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Read_RT_Status_Px

Description Read_RT_Status_Px returns 1 if a 1553 Message has been received since the last time Read_RT_Status_Px was called.

Syntax Read_RT_Status_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
status	1 Message was received 0 No message was received

Read_RTid_Px

Description	Read_RTid_Px returns an array of 32 words that are found in the data block assigned to the selected RTid, selecting the appropriate buffer of the double-buffer set.	
Syntax	Read_RTID_Px (`int` handle, `int` rtid, `usint` *words)	
Input Parameters	handle	The handle designated by Init_Module_Px
	rtid	11 bit identifier including RT#, receive bit and subaddress. See RT Identifier on page 3-2 and RT_Id_Px on page 3-16.
Output Parameters	words	Pointer to an array of 32 words. Although the message may contain fewer than 32 words, 32 words will be transferred.
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	einval	If parameter is out of range
	ercvfunc	If T/R bit is set to TRANSMIT
	edbnotset	If selected RTid does not have double-buffering set
	0	If successful

Reset_RT_Interrupt_Px

Description	Reset_RT_Interrupt_Px turns off request for Message Complete interrupt. A Message Complete interrupt will not be generated following this call until Set_RT_Interrupt_Px is called.	
Syntax	Reset_RT_Interrupt_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	0	If successful

RT_Id_Px

Description	RT_Id_Px is a utility function to help the user calculate the RTid value. (See **RT Identifier** on page 3-2.)		
Syntax	RT_Id_Px (`int` rtnum, `int` type, `int` subaddr, `int` *rtid)		
Input Parameters	rtnum	Address of the RT Valid values: 0 – 31	
	type	TRANSMIT	Allocate memory for data to be transmitted [0001 H]
		RECEIVE	Allocate memory for data to be received [0000 H]
	subaddr	Designated Subaddress of the given RT Valid values: 0 – 31	
Output Parameters	rtid	The calculated RTtid	
Return Values	einval	If an invalid value or parameter was used as an input	
	0	If successful	

Run_RT_Px

Description	Run_RT_Px starts RT simulation for all RTs that have been activated with Set_RT_Active_Px.	
Syntax	Run_RT_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	0	If successful

Set_1553Status_Px

Description Set_1553Status_Px provides a value, with a flag specifying duration, to be transmitted as the 1553 Status Word for the specified RT.

Syntax Set_1553Status_Px (`int` handle, `int` rtnum, `int` statusvalue, `int` duration)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtnum	Address of the RT Valid values: 0 – 31	
statusvalue	1553 Status word value to be transmitted	
duration	DUR_ALWAYS	The 1553 Status word specified should *always* be used for the RT [0000 H]
	DUR_ONETIME	The 1553 Status word specified should be used only *once* for this RT [0001 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If parameter is out of range
emode	If module is not in RT mode
0	If successful

Set_Bit_Px

Description Set_Bit_Px provides a value to be returned in response to the Transmit BIT (Built in Test) Word mode command for the given RT.

Syntax Set_Bit_Px (`int` handle, `int` rtnum, `int` bitvalue)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	Address of the RT Valid values: 0 – 31
bitvalue	BIT response value – the value to be set

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_Bit_Cnt_Px

Description Set_Bit_Cnt_Px determines how many bits will be sent for messages that have requested BIT_CNT_ERR errors. The number sent will be the legal number of bits in a word (20) + offset.

Syntax Set_Bit_Cnt_Px(`int` handle, `int` offset)

Input Parameters

handle	The handle designated by Init_Module_Px
offset	Number of bits to add or subtract from the correct number of bits in a 1553 word (20). Valid values: -3 to +3

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_Broad_Interrupt_Px

Description Set_Broad_Interrupt_Px determines whether an interrupt is to be generated when a broadcast message is detected.

Syntax Set_Broad_Interrupt_Px (`int` handle, `int` intrpt)

Input Parameters

handle	The handle designated by Init_Module_Px	
intrpt	ENABLE	Generate an interrupt [0001 H]
	DISABLE	Do *not* generate an interrupt {0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_Checksum_Blocks_Px

Description Set_Checksum_Blocks_Px is used to specify which RTid will receive a checksum. Associate an RTid with a data block using Assign_RT_Data_Px, see page 3-5. RTids associated with all data blocks numbered up to the value specified in this function will receive a checksum.

The value set in this function is returned by Get_Checksum_Blocks_Px, see page 3-6.

Note: This function is applicable *only* for modules with the 1760 options.

Syntax Set_Checksum_Blocks_Px (`int` handle, `int` csum_blocks)

Input Parameters

handle	The handle designated by Init_Module_Px	
csum_blocks	Number of data blocks to receive checksum. Valid values: 1 – 200 For example: A value of 5 means that data blocks 0 – 4 will receive a checksum; 5 – 199 will not.	

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
<0	If failure
0	If successful

Set_Invalid_Data_Res_Px

Description Set_Invalid_Data_Res_Px determines whether RTs are to respond to messages containing invalid data words.

Syntax Set_Invalid_Data_Res_Px (`int` handle, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px	
flag	ENABLE	Respond to message with invalid data [0001 H]
	DISABLE	Do *not* respond to this message with invalid data [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_Mode_Addr_Px

Description

Set_Mode_Addr_Px defines which subaddresses in a Command Word are to be used to indicate that the command in a Mode Command, *instead* of being interpreted as an RT subaddress.

Syntax

Set_Mode_Addr_Px (`int` handle, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px
flag	'0' 11111 and 00000 are mode commands '1' only 00000 is a mode command '2' only 11111 is a mode command

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_RT_Active_Px

Description

Set_RT_Active_Px causes a particular RT to be simulated. When an RT is simulated the 1553 Data words are *either* transmitted from an assigned data block *or* received by the RT and are then stored in the assigned data block.

See Assign_RT_Data_Px, on page 3-5, to assign a data block.

Input parameter, intrpt, lets the user request an interrupt when a message is processed for every RTid associated with this RT.

This function may be called in BC/Concurrent-RT mode to turn on Concurrent RT simulation.

Syntax

Set_RT_Active_Px (`int` handle, `int` rtnum, `int` intrpt)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	Address of the RT Valid values: 0 – 31
intrpt	1 Generate 'message complete' interrupt when a message is processed by this RT 0 Do *not* generate interrupt

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_RT_Active_Bus_Px

Description Set_RT_Active_Bus_Px causes a particular RT to respond to a specified bus or buses.

Syntax Set_RT_Active_Bus (int handle, int rtnum, usint bus)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtnum	Address of the RT Valid values: 0 – 31	
bus	BUS_A_ONLY	Bus A only [0002 H]
	BUS_B_ONLY	Bus B only [0001 H]
	BUS_AB	Buses A and B [0000 H]
	BUS_NONE	No buses active [0003 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_RT_Broadcast_Px

Description Set_RT_Broadcast_Px designates RT address 31 as either the broadcast address or as a regular RT.

Syntax Set_RT_Broadcast_Px (int handle, int toggle)

Input Parameters

handle	The handle designated by Init_Module_Px	
toggle	ENABLE	Enable RT31 as the broadcast address [0001 H]
	DISABLE	Disable RT31 from being the broadcast address [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
0	If successful

Set_RT_Errors_Px

Description Set_RT_Errors_Px sets up which errors are to be inserted by all RTs.

Syntax Set_RT_Errors_Px (`int` handle, `int` errormask)

Input Parameters

handle	The handle designated by Init_Module_Px	
errormask	All requested errors may be ORed together. The following errors are defined:	
	BIT_CNT	Insert bad bit count based on Set_Bit_Cnt_Px [0002 H]
	BAD_GAP_TIME	Insert bad gap between Data words 1 and 2 [0004 H]
	STATUS_PARITY	Send Status word with even parity [0010 H]
	STATUS_SYNC	Send Status word with data sync [0020 H]
	DATA_PARITY	Send Data word with even parity [0040 H]
	DATA_SYNC	Send Data word with command sync [0080 H]
	0	Clears error injection: no errors are inserted by any RT on a transmit message [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_RT_Interrupt_Px

Description Set_RT_Interrupt_Px has been retained for backward compatibility. Use Set_Interrupt_Px on page 2-16.

Syntax Set_RT_Interrupt_Px (`int` handle)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	0	If successful

Set_RT_Nonactive_Px

Description Set_RT_Nonactive_Px sets a particular RT address nonactive, thereby turning off simulation for the specified RT. The module will not respond to messages sent to nonactive RTs nor will it store messages to or from nonactive RTs.
This function may be called in BC mode to turn off concurrent RT simulation.

Syntax Set_RT_Nonactive_Px (`int` handle, `int` rtnum)

Input Parameters	handle	The handle designated by Init_Module_Px
	rtnum	Address of the RT Valid values: 0 – 31
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	0	If successful

Set_RT_Resp_Time_Px

Description Set_RT_Resp_Time_Px sets response time for RTs. This is the time between receipt of the command (and data for BC-to-RT messages) and the transmission of the 1553 Status word by the RT.

This function may be called in BC mode to adjust the timing of simulated concurrent RTs.

Syntax Set_RT_Resp_Time_Px (`int` handle, `int` nsecs)

Input Parameters

handle	The handle designated by Init_Module_Px
nsecs	Time in nanoseconds. Valid values: 4000 – 42000

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
0	If successful

Set_RTid_Interrupt_Px

Description Set_RTid_Interrupt_Px enables or disables the generation of an interrupt for the given RTid upon specified conditions.

Syntax Set_RTid_Interrupt_Px (`int` handle, `int` rtid, `int` enable, `int` int_type)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier on page 3-2 and RT_Id_Px on page 3-16.	
enable	ENABLE	Generate interrupt for this RTid [0001 H]
	DISABLE	Do not generate interrupt for this RTid [0000 H]
int_type	type of interrupt to generate	
	INT_ON_ERR	Generate interrupt on error [0001 H]
	INT_ON_ENDOFMSG	Generate interrupt on end of message [000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_RTid_Status_Px

Description Set_RTid_Status_Px enables or disables the ILLEGALIZATION or INACTIVE status of a given RTid.

Syntax Set_RTid_Status_Px (`int` handle, `int` rtid, `int` enable, `int` status_type)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier on page 3-2 and RT_Id_Px on page 3-16.	
enable	ENABLE	Enable the feature for this RTid [0001 H]
	DISABLE	Disable the feature for this RTid [0000 H]
status_type	Status to set for the RTid:	
	RTID_ILLEGAL	RTid will accept data on receive message, will not send data for transmit messages [0002 H]
	RTID_INACTIVE	RTid will not respond to any messages sent to it [0003 H]

Note: For Transmit messages with RTID_INACTIVE enabled, in order to ensure that the RT does not send the Status word back to the BC, the RT Response Time *must* be set to a minimum of 5μsec. The default setting is 4 μsec.

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_Status_Px

Description	Set_Status_Px provides a value to be transmitted as the 1553 Status word for the specified RT.	
Syntax	Set_Status_Px (`int` handle, `int` rtnum, `int` statusvalue)	
Input Parameters	handle	The handle designated by Init_Module_Px
	rtnum	Address of the RT Valid values: 0 – 31
	statusvalue	1553 Status word value to be transmitted
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	einval	If parameter is out of range
	0	If successful

Set_Var_Amp_Px

Description	Set_Var_Amp_Px enables the user to set the amplitude of the 1553 output signal.	
Syntax	Set_Var_Amp_Px (`int` handle, `int` millivolts)	
Input Parameters	handle	The handle designated by Init_Module_Px
	millivolts	Amplitude in millivolts. Valid range: 0 – 7.5 volts Value is rounded up to the nearest 30 millivolts.
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	einvamp	If tried to set an invalid amplitude
	0	If successful

Set_Vector_Px

Description Set_Vector_Px provides a value to be transmitted in response to the Transmit Vector Word mode command for the specified RT.

Syntax Set_Vector_Px (`int` handle, `int` rtnum, `int` vecvalue)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	Address of the RT Valid values: 0 – 31
vecvalue	Service Request vector response value to be set

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_Wd_Cnt_Err_Px

Description Set_Wd_Cnt_Err_Px requests the selected RT to insert a Word Count Error in its transmissions. Each individual RT can be set to send up to three words more than the word count or up to three words less than the word count.

Syntax Set_Wd_Cnt_Err_Px(`int` handle, `int` rtnum, `int` offset)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	Address of the RT Valid values: 0 – 31
offset	Offset to add to correct word count. Valid values: -3 to +3

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

4 Bus Monitor Functions

Chapter 4 describes the Excalibur P*x* module and *PCMCIA/EPII* operation in Bus Monitor mode. The Bus Monitor mode is used to simulate the Bus Monitor in an application.

Any reference to a module, also applies to the *EXC-1553PCMCIA/EPII* card.

Galahad Software Tools supports two different submodes for Monitoring:

- Look Up Submode is useful for applications in which the last data for a given RTid is important. In this mode all messages to the same RTid are stored in the same buffer.
- Sequential Submode is used when the sequence of the incoming messages is important. In this mode messages are stored sequentially in a circular buffer. This submode supports extensive filtering and triggering capabilities which causes the module to record only wanted messages and saves valuable processing time.

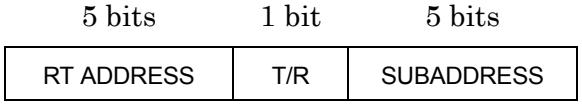
In each mode, the user can select which messages will cause the module to generate an interrupt.

The functions described in this chapter are listed, under their submode. Functions that can be used with both submodes are listed under **All Submodes**.

All Submodes	Look Up Submode	Sequential submode
Clear_Msg_Blks_Px	Assign_Blk_Px	Get_Counter_Px
Get_MON_Status_Px	Enable_Lkup_Int_Px	Get_Next_Message_Px
Run_MON_Px	Get_Last_Blknum_Px	Set_Cnt_Trig_Px
Set_Broad_Ctl_Px	Get_Message_Px	Set_Trigger_Mode_Px
Set_Mode_Addr_Px		Set_Trigger1_Px
Set_MON_Concurrent_Px		Set_Trigger2_Px
Set_Mon_Response_Time_Px		

RT Identifier

Many of the functions in this section take an RT identifier (RTid) as an argument. The RTid is defined as an RT address, T/R bit value and RT subaddress combination. The structure of the RTid is illustrated below:



Example: RT 5, Transmit, Subaddress 6 would be represented as 00101 1 00110 or 0166 H. This value can be isolated from a Command word by shifting the Command word 5 bits to the right.

The RT_Id_Px function (on page 3-16) is provided to carry out this calculation.

All Submodes

Clear_Msg_Blks_Px

Description	Clear_Msg_Blks_Px clears out all 1553 messages previously received and stored in memory. The Monitor Status register is also cleared. If the module is running, it is stopped, cleared and restarted.	
Syntax	Clear_Msg_Blks_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in Monitor mode
	0	If successful

Get_MON_Status_Px

Description Get_MON_Status_Px returns the status of the Monitor. The function is useful to determine what condition triggered an interrupt.
Note: The status is automatically reset each time it is read.

Syntax Get_MON_Status_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values Status word to check for the following flags:

MSG_IN_PROGRESS	A message is in the process of being received [0002 H]
TRIG_RCVD	A message which matched trigger1 or trigger2 has been received [0001 H]
CNT_TRIG_MATCH	The number of messages recorded by the module has reached the number set by Set_Cnt_Trig_Px [0004 H]
ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode
0	If successful

Run_MON_Px

Description Run_MON_Px causes the Monitor module to start.

Syntax Run_MON_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode
etimeout	If Monitor is currently running
0	If successful

Set_Broad_Ctl_Px

Description Set_Broad_Ctl_Px designates the RT address 31 as either the broadcast address or as a regular RT.

Syntax Set_Broad_Ctl_Px (`int` handle, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px	
flag	ENABLE	Sets RT31 as the broadcast address [0001 H]
	DISABLE	Sets RT31 as a regular RT [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode
einval	If an invalid parameter was used as an input
0	If successful

Set_Mode_Addr_Px

Description Set_Mode_Addr_Px defines which 'subaddresses' in a Command Word are to be used to indicate that the command in a Mode Code, *instead* of being interpreted as an RT subaddress.

Syntax Set_Mode_Addr_Px (`int` handle, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px
flag	'0' 11111 and 0000 are mode commands '1' only 00000 is a mode command '2' only 11111 is a mode command

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode
einval	If an invalid parameter was used as an input
0	If successful

Set_MON_Concurrent_Px

Description

Set_MON_Concurrent_Px links pairs of modules to each other such that one module of the pair can monitor the other module in the pair internally *without* attaching to the bus.

PCI and **cPCI**	Module 1 can monitor module 0 and module 3 can monitor module 2 without attaching to the bus.
VME and **VXI**	In addition to the above, module 5 can monitor module 4 and module 7 can monitor module 6 without attaching to the bus.
PCMCIA/EPII	Not applicable

Syntax

Set_MON_Concurrent_Px (`int` handle, `int` enable)

Input Parameters

handle	The handle designated by Init_Module_Px	
enable	ENABLE	Enable the internal monitor links [0001 H]
	DISABLE	Disable the internal monitor links [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
econcurrmonmodule	If module 1 or 3 (or modules 5 or 7 on VME/VXI boards) is not currently selected
einval	If an invalid parameter was used as an input
func_invalid	If called for a *PCMCIA/EPII* card
0	If successful

Set_Mon_Response_Time_Px

Description	Set_Mon_Response_Time_Px sets the Monitor's response time window. The value determines the maximum wait time until an RT's Status Response is considered invalid by the Monitor.	
Syntax	Set_Mon_Response_Time_Px (`int` handle, `usint` rtime)	
Input Parameters	handle	The handle designated by Init_Module_Px
	rtime	Permissible response time in microseconds
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in Bus Monitor mode
	0	If successful

Look Up Submode

Assign_Blk_Px

Description Assign_Blk_Px assigns a data block to a RT subaddress.

Syntax Assign_Blk_Px (`int` handle, `int` rtid, `int` blknum)

Input Parameters

handle	The handle designated by Init_Module_Px
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier on page 4-2 and RT_Id_Px on page 3-16.
blknum	Block number: Valid values: 1 – 127 Assign data block to RTid *or* 0 De-assigns block‘

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode
einval	If an invalid parameter was used as an input
0	If successful

Enable_Lkup_Int_Px

Description Enable_Lkup_Int_Px enables or disables interrupts for the specified RTid.

Syntax Enable_Lkup_Int_Px (`int` handle, `int` rtid, `int` toggle)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier on page 4-2 and RT_Id_Px on page 3-16.	
toggle	ENABLE	Enable interrupts for the RTid [0001 H]
	DISABLE	Disable interrupts for the RTid [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If an invalid parameter was used as an input
0	If successful

Get_Last_Blknum_Px

Description Get_Last_Blknum_Px returns the number of the last data block written to.

Syntax Get_Last_Blknum_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode
otherwise	Number of the current data block in use: Valid values 1 – 127
0	No block

Get_Message_Px

Description Get_Message_Px retrieves a specified message from the Message Block Area.

Syntax Get_Message_Px (`int` handle, `int` blknum, `struct MONMSG` *msgptr)

Input Parameters

handle	The handle designated by Init_Module_Px
blknum	The blknum for the most recent message is obtained via the Get_Last_Blknum_Px function. Valid values: 0 – 127

Output Parameters

msgptr	Pointer to the structure defined below with the data returned. Space should always be allocated for 36 words of data to accommodate the maximum case of RT-to-RT transmission of 32 Data words + 2 Status words + 2 Command words.

typedef struct {

int msgstatus; Status word containing the following flags:

END_OF_MSG	Indicates end of message [8000 H]
TRIGGER_FOUND	Trigger message was received and restored [4000 H]
RT2RT_MSG	Message was RT-to-RT transfer [2000 H]
ME_SET	Message Error bit in the RT Status word [1000 H]
BAD_STATUS	RT Status word bits set (not ME bit) [0800 H]
INVALID_MSG	Word count or Sync error occurred [0400 H]

Get_Message_Px (cont.)

BM_BAD_CHECKSUM	Bad checksum [0200 H] (1760 options only)
BUS_A_XFER	Message was transferred on bus A [0100 H]
INVALID_WORD	Bad bit count, Manchester or parity [0080 H]
BAD_HEADER_MON*	Header error [0060 H] (for 1760 options only)

Note: If 1760 options are in effect, set *either* bits 05 *or* 06 for Word Count Errors; both bits must be set for Header Error.

WORD_CNT_HI	RT transmitted too many words [0040 H]
WORD_CNT_LO	RT transmitted too few words [0020 H]
BAD_RT_ADDR	Received 1553 Status word did not contain the correct RT address [0010 H]
BAD_SYNC	Sync of either the Command or the Data word(s) is incorrect [0008 H]
BAD_GAP	Invalid gap received between 1553 Words [0004 H]
MON_LATE_RESP	Response time error occurred in the message, even if no RT active on the module [0002 H]
MSG_ERROR	Error occurred, defined in other flags [0001 H]

unsigned long elapsedtime;	The 32-bit Time tag associated with the message.
int *words;	A pointer to an array of 1553 words.

}

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode
einval	If an invalid parameter was used as an input
0	If successful

Sequential Submode

Get_Counter_Px

Description Get_Counter_Px returns the number of the last block to be filled with a monitored message. There are 200 blocks, numbered 0 – 199.

Syntax Get_Counter_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Sequential Monitor mode
counter	0 – 199

Get_Next_Message_Px

Description Get_Next_Message_Px reads the message block following the block read in the previous call to Get_Next_Message_Px. The first call to Get_Next_Message_Px will return block 0.

Note: If the next block does not contain a new message an error will be returned. If the user falls 200 blocks behind, it is impossible to know if the message returned is newer or older than the previous returned message unless the Time tag is checked. The function therefore returns the Time tag itself in the elapsed time field rather than a delta value.

Syntax Get_Next_Message_Px (`int` handle, `struct MONMSG` *msgptr)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

msgptr	Pointer to the structure defined below in which to return the message

struct MONMSG {

int msgstatus;	Status word containing the following flags:
END_OF_MSG	Indicates end of message [8000 H]
TRIGGER_FOUND	Trigger message was received and restored [4000 H]
RT2RT_MSG	Message was RT-to-RT transfer [2000 H]
ME_SET	Message Error bit in the RT Status word [1000 H]

Get_Next_Message_Px (cont.)

BAD_STATUS	RT Status word bits set (not ME bit) [0800 H]
INVALID_MSG	Word count or Sync error occurred [0400 H]
BM_BAD_CHECKSUM	Bad checksum [0200 H] (1760 options only)
BUS_A_XFER	Message was transferred on bus A [0100 H]
INVALID_WORD	Bad bit count, Manchester or parity [0080 H]
BAD_HEADER_MON*	Header error[0060 H] (for 1760 options only)

Note: If 1760 options are in effect, set *either* bits 05 *or* 06 for Word Count Errors; both bits must be set for Header Error.

WORD_CNT_HI	RT transmitted too many words [0040 H]
WORD_CNT_LO	RT transmitted too few words [0020 H]
BAD_RT_ADDR	Received 1553 Status word did not contain the correct RT address [0010 H]
BAD_SYNC	Sync of either the Command or the Data word(s) is incorrect [0008 H]
BAD_GAP	Invalid gap received between 1553 Words [0004 H]
MON_LATE_RESP	Response time error occurred in the message, even if no RT active on the module [0002 H]
MSG_ERROR	Error occurred, defined in other flags [0001 H]

unsigned long elapsedtime;	The 32-bit Time tag associated with the message.
unsigned int *words [36];	A pointer to an array of 1553 words in the sequence they were received over the bus.

}

Get_Next_Message_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in Monitor mode
	enomsg	If there is no message to return
	eoverrun	If message was overwritten in the middle of being read here. At least 200 messages were received since the previous Get_Next_Message_Px, so the Time tags are out of sync.
	block number	If successful: valid values 0 – 199

Set_Cnt_Trig_Px

Description	Set_Cnt_Trig_Px sets a block number as the counter trigger. When this block is filled, the CNT_TRIG_MATCH bit within the Monitor status word will be set and, if requested via the Set_Interrupt_Px function an interrupt will be generated.	
Syntax	Set_Cnt_Trig_Px (`int` handle, `int` blknum)	
Input Parameters	handle	The handle designated by Init_Module_Px
	blknum	Block numbers Valid values: 0 – 199
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in Sequential Monitor mode
	einval	If an invalid parameter was used as an input
	0	If successful

Set_Trigger_Mode_Px

Description

Set_Trigger_Mode_Px determines:

1. The action to be taken upon receiving a message matching the trigger.
2. If the source of the trigger will be the Command word or the message Status Word.

The trigger message is always stored, and the user can choose to store all messages from this point on or only those messages matching the trigger.

To disable the triggers, call Set_Trigger_Mode_Px with a mask argument of 0000.

Note: Set_Trigger_Mode_Px must be called *after* Set_Trigger1_Px and Set_Trigger2_Px. Otherwise, the mode may not be set properly.

Syntax

Set_Trigger_Mode_Px (`int` handle, `int` mode)

Input Parameters

handle		The handle designated by Init_Module_Px	
mode		Combination of one flag from each group ORed together	
	When to trigger	STORE_AFTER	Default – Causes storage of messages to begin following the first occurrence of a trigger message [0010 H]
		STORE_ONLY	Causes only messages matching the trigger to be stored [0008 H]
	On what to trigger	TCOMMAND	Default – Causes triggering to be based on the Command word [0000 H]
		STATUS_TRIGGER	Causes triggering to be based on the Message Status word [0080 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Sequential Monitor mode
einval	If an invalid parameter was used as an input
0	If successful

Set_Trigger1_Px

Description

In Set_Trigger_Mode_Px the user can select a 1553 Command word or a Message Status word for the monitor to trigger on. (See Set_Trigger_Mode_Px on page 4-13.)

In Set_Trigger1_Px the user specifies which Control word or Message Status word is the trigger. When a message matching this parameter is encountered, the trigger is met and the action specified in Set_Trigger_Mode_Px is taken.

The user has the option of setting an additional trigger using Set_Trigger2_Px.

Note: Set_Trigger1_Px must be called *before* Set_Tigger_Mode_Px. Otherwise, the mode may not be set properly.

Syntax

Set_Trigger1_Px (`int` handle, `int` trigger, `int` mask)

Input Parameters

handle	The handle designated by Init_Module_Px
trigger	16 bit trigger value – a Command Word or Message Status Word See Set_Trigger_Mode_Px on page 4-13
mask	Mask for trigger. Specifies which bits in the trigger word are required to match and which are 'don't care' '1' in required bit positions '0' in don't care bit positions Mask of all '0's turns off trigger
Example	When triggering on Command words, Set_Trigger1_Px or Set_Trigger2_Px (0843 H, F800 H) would trigger all messages to/from RT1.

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Sequential Monitor mode
0	If successful

Set_Trigger2_Px

Description

In Set_Trigger_Mode_Px the user can select a 1553 Command word or a Message Status word for the monitor to trigger on. (See Set_Trigger_Mode_Px on page 4-13.)

In Set_Trigger2_Px the user specifies which Control word or Message Status word is the trigger. When a message matching this parameter is encountered, the trigger is met and the action specified in Set_Trigger_Mode_Px is taken.

The user has the option of setting an additional trigger using Set_Trigger1_Px.

Note: Set_Trigger2_Px must be called *before* Set_Tigger_Mode_Px. Otherwise, the mode may not be set properly.

Syntax

Set_Trigger2_Px (`int` handle, `int` trigger, `int` mask)

Input Parameters

handle	The handle designated by Init_Module_Px
trigger	16 bit trigger value – a Command Word or Message Status Word See Set_Trigger_Mode_Px on page 4-13
mask	Mask for trigger. Specifies which bits in the trigger word are required to match and which are 'don't care' '1' in required bit positions '0' in don't care bit positions Mask of all 0s turns off trigger
Example	When triggering on Command words, Set_Trigger1_Px or Set_Trigger2_Px (0843 H, F800 H) would trigger all messages to/from RT1.

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Sequential Monitor mode
0	If successful

5 BC/Concurrent-RT Functions

Chapter 5 describes the *Galahad Software Tools* functions for operating the P*x* module and *PCMCIA/EPII* card in BC/Concurrent-RT mode. To set the module in BC/Concurrent-RT mode, see Set_Mode_Px on page 2-17.

In BC/Concurrent-RT mode the module operates as a Bus Controller. The user can define 1553 messages and group them into frames and has control over which commands and data are transmitted over the bus as well as the timing between messages and frames. Error injection options permit detailed testing of RTs, and error detection features provide rich reporting capabilities.

In addition, the module can simulate up to 32 Remote Terminals using the function Set_RT_Active_Px on page 5-38.

Any reference to a module, also applies to the *EXC-1553PCMCIA/EPII* card.

The following functions are described in this chapter:

Alter_Cmd_Px
Alter_IMG_Px
Alter_Message_Px
Alter_MsgSendTime_Px
BC_Check_Alter_Msgentry_Px
Clear_Card_Px
Clear_Frame_Px
Clear_Timetag_Sync_Px
Command_Word_Px
Create_1553_Message_Px
Create_Frame_Px
Enable_Checksum_Px
Enable_Checksum_Error_Px
Enable_SRQ_Support_Px
Get_BC_Msgentry_Px
Get_BC_Status_Px
Get_Minor_Frame_Time_Px
Get_Next_Message_BCM_Px
Insert_Msg_Err_Px
Re_Create_Message_Px
Read_Message_Px
Reset_BC_Status_Px
Run_BC_Px
Select_Async_Frame_Px
Send_Async_Frame_Px
Send_Timetag_Sync_Px
Set_BC_Resp_Px
Set_Bus_Px
Set_Continue_Px
Set_Error_Location_Px
Set_Frame_Time_Px
Set_Halt_Px
Set_Interrupt_On_Msg_Px
Set_Jump_Px
Set_Minor_Frame_Time_Px
Set_Replay_Px
Set_Restore_Px
Set_Retry_Px
Set_RT_Active_Px
Set_RT_Nonactive_Px
Set_RT_Resp_Time_Px
Set_Skip_Px
Set_Stop_On_Error_Px
Set_Sync_Pattern_Px
Set_Var_Amp_Px
Set_Word_Cnt_Px
Set_Zero_Cross_Px
Start_Frame_Px

The input values included in each function are given, in Hex format, by each flag within the function description.

BC/Concurrent-RT Simulation

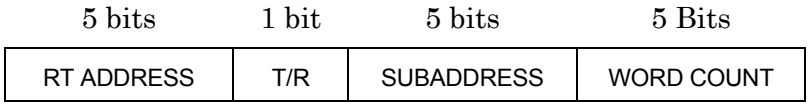
In BC/Concurrent-RT mode, the Create_1553_Message_Px function is used to define both the BC message and any simulated RT response. In BC/Concurrent-RT mode, the maximum number of Words in the message is 36:

	Maximum number of Data words	32
+	Maximum number of Command words	2
+	Maximum number of Status words	2
	Maximum number of Words in the message	36

Refer to the hardware *User's Manual* that came with your module for further explanation and examples of this mode.

Command Word Calculation

Some of the functions in BC/Concurrent-RT mode take the 1553 Command word as an argument. The structure of the Command word is:



Example: RT5, Receive, Subaddress 6 and Word count 15 would be represented as 0010 1000 1100 1111 in binary or 28CF in hex.

The Command_Word_Px function, on page 5-12, is provided to carry out this calculation.

Servicing the Service Request (SRQ) Bit

The SRQ bit is set for a Remote Terminal (RT) in the 1553 RT Status Word. Setting the SRQ bit indicates to the Bus Controller (BC) that the RT/Subaddress requires service.

In response, the BC provides the following service: the module will send out a mode code 16 (transmit Vector Word) to get the Vector Word from the RT which contains more information about what needs service. The BC will then send out a transmit message to the Subaddress (provided that it is not set to 0) identified in the Vector Word.

Functions by Category

The more frequently called functions are grouped according to use. Page references are given only if the functions do not appear in this chapter.

Sending out a fixed number of messages in a loop

Init_Module_Px for PCI boards and PCMCIA cards	To initialize the module/card (page 2-8)
Set_Mode_Px	To place the module in BC/Concurrent-RT mode (page 2-17)
Create_1553_Message_Px	To create the messages to send
Create_Frame_Px	To select the order in which to transmit the messages
Set_Frame_Time_Px	To determine the time between transmission of frames
Start_Frame_Px	To choose the frame to transmit
Run_BC_Px	To start the module running
Stop_Px	To Stop the RT
Release_Module_Px	To release resources assigned to the module

Additional frame functions:

Select_Async_Frame_Px	To designate a frame to be sent out by Send_Async_Frame
Send_Async_Frame_Px	To send the asynchronous frame that was setup by a call to Select_Async_Frame
Set_Minor_Frame_Time_Px	To set the minor frame time for a Minor frame type of message

Obtaining data and status information during transmission

Read_Message_Px	To get data and 1553 status information
Get_BC_Msgentry_Px	To get both data and 1553 status information AND status information for a particular message
Get_BC_Status_Px	To get module level status information
Reset_BC_Status_Px	To clear the BC status
Get_Next_Message_BCM_Px	To read messages monitored on the bus by the Internal Concurrent Monitor

Changing message parameters in real-time

Set_BC_Resp_Px	To select how long to wait for an RT to respond
Set_Halt_Px	To stop transmission at a selected message
Set_Continue_Px	To resume transmission following a halt
Set_Skip_Px	To skip transmission of a specified message
Set_Restore_Px	To restore a skipped message
Set_Stop_On_Error_Px	To cause the module to stop if it detects an error
Clear_Card_Px	To fully clear (entire module) in order to restart
Clear_Frame_Px	To partially clear (only frame) in order to restart

Setting RTs

In BC/Concurrent-RT mode, the user can simulate an entire system by simulating one or more RTs in addition to simulating the Bus Controller. To simulate RTs, set up the Data and Status Words that the simulated RTs are expected to send in Create_1553_Message_Px and Alter_Message_Px.

The functions to simulate RTs are described in detail in **Chapter 3: Remote Terminal Functions**.

Set_RT_Active_Px	To select the RTs to be simulated	page 5-38
Set_RT_Nonactive_Px	To turn off simulation of a RT	page 5-38
Set_RT_Resp_Time_Px	To select how long to wait before responding to a command	page 3-24

Note: These RT functions are the only ones usable in BC/Concurrent-RT mode. Use Run_BC_Px to start BC/Concurrent-RT module operation and *not* Run_RT_Px.

To send checksum values – for 1760 options modules

With the Excalibur P*x* modules which support the 1760 options in BC and BC/Concurrent-RT modes, the user can send a checksum value in place of the last Data Word in the message. In addition, an error may be injected into this checksum value. The following functions control these functions:

Enable_Checksum_Px	To select whether or not to place checksum as the last Data Word of a message
Enable_Checksum_Error_Px	To select whether or not to use checksum error injection on the message

BC/Concurrent-RT Functions

Alter_Cmd_Px

Description Alter_Cmd_Px changes the Command word of a previously defined message. The function takes an RT address and a SA as arguments and copies these into the original Command word. The number of Data words cannot be changed.

Syntax Alter_Cmd_Px (`int` handle, `int` id, `usint` rt, `usint` sa)

Input Parameters

handle	The handle designated by Init_Module_Px	
id	Message identifier returned from a prior call to Create_1553_Message_Px	
rt	The new RT number Valid values: 0 – 31 *or*	
	SAME_RT	Do not change the RT of the Command [0020 H]
sa	The new Subaddress number Valid values: 0 – 31 *or*	
	SAME_SA	Do not change the SA of the Command [0020 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC mode
ebadid	If id is not a valid message id
0	If successful

Alter_IMG_Px

Description

Alter_IMG_Px changes the intermessage gap time of a previously defined message.

Syntax

Alter_IMG_Px (`int` handle, `int` frameid, `int` msgentry, `unsigned long` img)

Input Parameters

handle	The handle designated by Init_Module_Px
frameid	Frame identifier returned from a prior call to Create_Frame_Px
msgentry	Entry within the frame, i.e., 0 for the first message in the frame, 1 for the second message, etc.
img	New intermessage gap time in microseconds

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
frm_erange	If frameid is not a valid frame id
frm_erangecnt	If msgentry is greater than the number of messages in the frame
0	If successful

Alter_Message_Px

Description

Alter_Message_Px changes the data of a previously defined message. The function receives as input a pointer to an array of Data words and copies these words into the original message. The Command word cannot be changed and therefore the number of Data words cannot be changed. Only the values of the Data words (and Status word(s) for simulated RTs in BC/Concurrent-RT mode) can be changed.

Note: See **Appendix A: MIL-STD-1553 Formats**

Syntax

Alter_Message_Px (`int` handle, `int` id, `usint` *data)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px
data	Pointer to an array of up to 34 words of new Data words (plus Status word(s) for BC/Concurrent-RT mode) for message id. **Note:** Space must be left for the Status word, even if the RT is not simulated.

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
ebadid	If id is not a valid message id
emode	If not in BC/Concurrent-RT mode
eminorframe	If message is Minor Frame type
0	If successful

Alter_MsgSendTime_Px

Description Alter_MsgSendTime_Px changes a message send time previously defined in Create_Frame_Px. This is for use with the replay feature (Set_Replay_Px on page 5-35) to specify exactly when a message will be sent out.

Syntax Alter_MsgSendTime_Px (`int` handle, `int` frameid, `int` msgentry, `unsigned int` mst)

Input Parameters

handle	The handle designated by Init_Module_Px
frameid	Frame identifier returned from a prior call to Create_Frame_Px
msgentry	Entry within the frame, i.e. 0 for the first message in the frame, 1 for the second message, etc.
mst	The Time tag value at which the message should be sent out in replay mode

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
frm_erange	If frameid is not a valid frame id
frm_erangecnt	If msgentry is greater than the number of messages in the frame
0	If successful

BC_Check_Alter_Msgentry_Px

Description

To insure data integrity, call BC_Check_Alter_Msgentry_Px before a call to Alter_Message_Px. The function verifies that a message that is to be altered is not in the process of being transmitted or received. Without this check, a message may be altered in the middle of its transmission.

After a successful call to this function the user has at least 20 μsec. to call Alter_Message_Px or Read_Message_Px.

Note: BC_Check_Alter_Msgentry_Px replaces the obsolete function BC_Check_Alter_Msg_Px. BC_Check_Alter_Msgentry_Px is more accurate when multiple frames are used.

Syntax

BC_Check_Alter_Msgentry_Px (`int` handle, `int` frameid, `int` msgentry)

Input Parameters

handle	The handle designated by Init_Module_Px
frameid	Frame identifier returned from a prior call to Create_Frame
msgentry	Entry within the frame, i.e. 0 for the first message in the frame, 1 for the second message, etc.

Output Parameters

none

Return Values

ebadhandle		If an invalid handle was specified; should be value returned by Init_Module_Px
emode		If module is not in BC/Concurrent-RT mode.
einval		If an invalid parameter was used as an input
frm_erange		If `frameid` is not a valid frame id
frm_erangecnt		If msgentry is greater than the number of messages in the frame
Successful if:	ALTER_MSG	If message is available to be altered [0001 H]
	DO_NOT_ALTER_MSG	If message *is not* available to be altered [0000 H]

Clear_Card_Px

Description Clear_Card_Px clears out all messages and message frames (and the *Software Tools* variables associated with them) from the memory of the current module. Use this function to delete old, unnecessary messages to make room for new messages. It may be called to undo calls to Create_1553_Message_Px and Create_Frame_Px.

Syntax Clear_Card_Px (`int` handle)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Clear_Frame_Px

Description Clear_Frame_Px clears out all message frames previously created via Create_Frame_Px while leaving messages created via Create_1553_Message_Px. May be used in situations when a limited number of messages must be reformulated into different frames in realtime.

Note: It is often more efficient to use the Set_Skip_Px and Set_Restore_Px functions for this situation.

Syntax Clear_Frame_Px (`int` handle)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Clear_Timetag_Sync_Px

Description

Clear_Timetag_Sync_Px resets the Time tag register to 0 when the BC sends out a Mode Code synchronize message. The function also operates in RT mode, when the RT receives the selected Mode Code sychronize message.

Syntax

Clear_Timetag_Sync_Px (`int` handle, `int` modecode, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px	
modecode	The selected Mode Code: 1 or 17	
flag	ENABLE	Reset Time tag at Mode Code 'Sync' [0001 H]
	DISABLE	Do not reset Time tag at Mode Code 'Sync' [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in BC/Concurrent-RT or RT mode
einval	If an invalid parameter was used as an input.
0	If successful

Command_Word_Px

Description Command_Word_Px is a utility function to help the user calculate the 1553 Command word value.

Syntax Command_Word_Px (`int` rtnum, `int` type, `int` subaddr, `int` wordcount, `usint` *commandword)

Input Parameters

rtnum	Address of the RT Valid values: 0 – 31	
type	TRANSMIT	Indicates that the RT/SA should transmit Data words [0001 H]
	RECEIVE	Indicates that the RT/SA should receive Data words [0000 H]
subaddr	Designated subaddress of the given RT Valid values: 0 – 31	
wordcount	The number of words to be received or transmitted in a message. Valid values:0 – 31 **Note:** For messages with at least one Data word, 0 indicates a word count of 32.	

Output Parameters

commandword	The calculated Command word

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If an invalid parameter was used as an input.
0	If successful

Create_Frame_Px

Description

Create_Frame_Px creates a frame of previously defined messages to be sent by the BC. Each element in the frame structure consists of a message id (the value returned from a call to Create_1553_Message_Px), and a gap time indicating the time between the transmission of the message and the following one. The frame structure must contain a final entry with id '0', indicating the end of the frame.

Up to 20 frames may be defined by calls to Create_Frame_Px, each call returning a frame id which can be used as a parameter to Start_Frame_Px to specify which frame to send next. Run_BC_Px then sends the messages in that frame.

Note: When setting up a frame for use in Replay mode, indicate the required 'message send time' for each message, in place of the gap time. (See Set_Replay_Px on page 5-35 and Alter_MsgSendTime_Px on page 5-8.)

Syntax

Create_Frame_Px (`int` handle, `struct FRAME` * framestruct)

Input Parameters

handle	The handle designated by Init_Module_Px

FRAME framestruct [ ];

struct FRAME {

short int id;	Message identifier returned by Create_1553_Message_Px or 0 for last message
long gaptime;	Gap Time between this message and following message in microseconds *or* Message Send Time – Time tag at which this message should be sent on in replay mode.

};

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
frm_badid	If tried to place an undefined message into a message frame
frm_nostack	If there is not enough space in frame stack for this frame
frm_maxframe	If exceeded maximum number of frames permitted (20)
frameid	If successful. A frame identifier of a frame just created. The frameid is used in many of the functions in BC/Concurrent-RT mode. Valid values 0 – 19

Create_1553_Message_Px

Description

Create_1553_Message_Px creates the basic building block of the 1553 protocol – the message. A message contains a minimum of one, and a maximum of two (for RT-to-RT messages), Command words and 32 Data words, depending on the type of message. Each message created with this function returns a unique id which is used to set or read characteristics of the message or to build a frame (i.e., a collection of messages) for subsequent transmission. The message must define the Data and Status words to be sent by the RT that is being simulated.

Note: See **Appendix A: MIL-STD-1553 Formats**

Syntax

Create_1553_Message_Px (`int` handle, `usint` cmdtype, `usint` data[ ], `short int` *id)

Input Parameters

handle	The handle designated by Init_Module_Px	
cmdtype	One of the following flags:	
	RT2BC	Send a transmit message [0000 H]
	BC2RT	Send a Receive message [0001 H]
	RT2RT	Send an RT to RT transfer message [0002 H]
	MODE	Mode Command [0003 H]
	BRD_RCV	Broadcast Receive message [0004 H]
	BRD_RT2RT	Broadcast RT to RT transfer message [0005 H]
	BRD_MODE	Broadcast Mode Command [0006 H]
	MINOR_FRAME	Minor frame message - delay [000F H]
data[ ]	A pointer to an array of command + data + status	

Note: In BC/Concurrent-RT mode the user may create an RT-to-RT message with 32 data words, 2 Command words and 2 Status words for 2 simulated RTs.

Output Parameters

id	On success, returns a message id identifying the message just created, for use in other function calls.

Create_1553_Message_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	ebadcommandword	If the T/R bit in the Command word is set incorrectly for the given cmdtype parameter
	einval	If an invalid parameter was used as an input
	msgnospace	If there is not enough space in message stack for this message
	msg2many	If exceeded maximum number of messages permitted (1660)
	0	If successful

Enable_Checksum_Px

Description Enable_Checksum_Px specifies if the last word of a message should be a checksum word or a regular Data word. This function is used for BC-to-RT and RT-to-RT messages.

Note: This function is applicable *only* to modules with 1760 options.

Syntax Enable_Checksum_Px (`int` handle, `int` frameid, `int` msgentry, `int` enable)

Input Parameters	handle	The handle designated by Init_Module_Px
	frameid	Frame identifier returned from a prior call to Create_Frame_Px
	msgentry	Entry within the frame, i.e. 0 for the first message in the frame, 1 for the second message, etc.
	enable	ENABLE Checksum is sent [0001 H]
		DISABLE Checksum is *not* sent [0000 H]
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	frm_erange	If frameid is not a valid frame id
	frm_erangecnt	If msgentry is greater than the number of messages in the frame
	0	If successful

Enable_Checksum_Error_Px

Description

Enable_Checksum_Error_Px sets checksum error injection for the message. Enable_Checksum_Px must be called for the message, to specify that checksum is sent.
This function is used for BC-to-RT and RT-to-RT messages.

Note: This function is applicable *only* to modules with 1760 options.

Syntax

Enable_Checksum_Error_Px (`int` handle, `int` frameid, `int` msgentry, `int` enable)

Input Parameters

handle	The handle designated by Init_Module_Px	
frameid	Frame identifier returned from a prior call to Create_Frame_Px	
msgentry	Entry within the frame, i.e. 0 for the first message in the frame, 1 for the second message, etc.	
enable	ENABLE	Erroneous checksum is sent [0001 H]
	DISABLE	Correct checksum is sent [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
frm_erange	If frameid is not a valid frame id
frm_erangecnt	If msgentry is greater than the number of messages in the frame
0	If successful

Enable_SRQ_Support_Px

Description

Enable_SRQ_Support_Px enables Service Request processing. See Servicing the Service Request (SRQ) Bit on page 5-2. If this function is not called, SRQ processing is enable by default.

Syntax

Enable_SRQ_Support_Px (`int` handle, `int` enableflag)

Input Parameters

handle	The handle designated by Init_Module_Px	
enableflag	ENABLE	Enable SRQ processing [0001 H]
	DISABLE	Disable SRQ processing [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC mode
einval	If an invalid parameter was used as an input
0	If successful

Get_BC_Msgentry_Px

Description

Get_BC_Msgentry_Px combines the status and the contents of a message in one structure, similar to the function of Get_Next_Message_RTM_Px (RT) and Get_Next_Message_BCM_Px (BC).

Get_BC_Msgentry_Px replaces the obsolete function Get_BC_Message_Px. Get_BC_Msgentry_Px is more accurate when multiple frames are used.

Syntax

Get_BC_Msgentry_Px (`int` handle, `int` frameid, `int` msgentry, `struct BCMSG` *msgptr)

Input Parameters

handle — The handle designated by Init_Module_Px

frameid — Frame identifier returned from a prior call to Create_Frame_Px

msgentry — Entry within the frame, i.e., 0 for the first message in the frame, 1 for the second message, etc.

Output Parameters

msgptr — Pointer to the structure defined below in which to return the data. Space should always be allocated for 36 words of data to accommodate the maximum case of RT-to-RT transmission of 32 Data words plus 2 Status words and 2 Command words.

```
struct BCMSG {
  unsigned int msgstatus
```

Status word containing the following words:

END_OF_MSG	Indicates end of message [8000 H]
BC_BAD_CHECKSUM	Bad checksum on the transmit side [4000 H]
BAD_BUS	RT response received on non-active channel [2000 H]
ME_SET	Message Error bit in the RT Status word [1000 H]
BAD_STATUS	Bit other than ME set in 1553 Status word [0800 H]
INVALID_MSG	Word count or Sync error occurred [0400 H]
LATE_RESP	RT responded late [0200 H]
BAD_HEADER_BC	Error in Header Word [0100 H] (for 1760 only)

Get_BC_Msgentry_Px (cont.)

	INVALID_WORD	Bad bit count or Manchester error or parity error [0080 H]
	WD_CNT_HI	RT transmitted too many words [0040 H]
	WD_CNT_LO	RT transmitted too few words [0020 H]
	BAD_RT_ADDR	1553 Status word contained wrong RT address [0010 H]
	BAD_SYNC	Status or Data sync were wrong [0008 H]
	BAD_GAP	Invalid Gap time occurred between words [0004 H]
	MSG_ERROR	Error occurred; defined in other flags [0001 H]
unsigned int words [36]	A pointer to an array of 1553 words.	
}		

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
einval	If an invalid parameter was used as an input
frm_erange	If `frameid` is not a valid frame id
frm_erangecnt	If msgentry is greater than the number of messages in the frame
0	If successful

Note: The function can be used instead of separately calling Get_Msgentry_Status_Px for the Status word, and then calling Read_Message_Px that returns just the content of the message.

Do *not* call Get_Msgentry_Status_Px before calling this function. An error message “enobcmsg” will be returned.

Get_BC_Status_Px

Description Get_BC_Status_Px indicates what condition triggered an interrupt. The function returns status flags generated since the last call to Reset_BC_Status_Px.

The function may be used for polling applications.

Note: Reset this status with the Reset_BC_Status_Px command.

Syntax Get_BC_Status_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
If successful	0 or more of the following flags:
MSG_ERR	An error in some message [0008 H]
END_OF_FRAME	Complete frame of messages sent [0004 H]
MSG_CMPLT	Message sent [0002 H]
WAIT_FOR_CONTINUE	Message named by Set_Halt_Px was reached [0001 H]

Get_Minor_Frame_Time_Px

Description Get_Minor_Frame_Time_Px gets the previously stored minor frame time in microseconds. This value was set in a call to Set_Minor_Frame_Time_Px.

Note: This function is valid only in BC/Concurrent-RT mode

Syntax Get_Minor_Frame_Time_Px (`int` handle, `long` *frame_time)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

frame_time	The minor frame time in microseconds

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
0	If successful

Get_Msgentry_Status_Px

Description

Get_Msgentry_Status_Px returns a Status word recorded by the Bus Controller associated with a selected message in a selected frame. If a valid message status is returned, the function clears the status.

Note: Get_Msgentry_Status_Px replaces the obsolete function Get_Msg_Status_Px. Get_Msgentry_Status_Px is more accurate when multiple frames are used.

Syntax

Get_Msgentry_Status_Px (`int` handle, `int` frameid, `int` msgentry, `usint` *msgstatusword)

Input Parameters

handle	The handle designated by Init_Module_Px
frameid	Frame identifier returned from a prior call to Create_Frame_Px
msgentry	Entry within the frame, i.e., 0 for the first message in the frame, 1 for the second message, etc.

Output Parameters

msgstatusword — On success, 0 or more of the following flags

END_OF_MSG	Complete 1553 message received [8000 H]
BC_BAD_CHECKSUM	Bad checksum on the transmit side [4000 H]
BAD_BUS	RT response received on non-active channel [2000 H]
ME_SET	Message Error bit in the RT Status word [1000 H]
BAD_STATUS	Bit other than ME set in 1553 Status word [0800 H]
LATE_RESP	RT responded late [0200 H]
BAD_HEADER_BC	Error in Header Word [0100 H] (for 1760 only)
INVALID_WORD	Bad bit count or Manchester error or parity error [0080 H]
WD_CNT_HI	RT transmitted too many words [0040 H]
WD_CNT_LO	RT transmitted too few words [0020 H]

Get_Msgentry_Status_Px (cont.)

	BAD_RT_ADDR	1553 Status word contained wrong RT address [0010 H]
	BAD_SYNC	Status or Data sync were wrong [0008 H]
	BAD_GAP	Invalid Gap time occurred between words [0004 H]
	MSG_ERROR	Error occurred; defined in other flags [0001 H]

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
einval	If an invalid parameter was used as an input
frm_erange	If called this function with an undefined frame
frm_erangecnt	If called to this function with a `msgentry` value greater than the number of messages in the frame
0	If successful

Get_Next_Message_BCM_Px

Description

Get_Next_Message_BCM_Px relates to the **Internal Concurrent Monitor** available in the memory of the module. The **Internal Concurrent Monitor** operates automatically when the module is started in BC/Concurrent-RT mode. All bus traffic in BC/Concurrent-RT mode is recorded in the **Internal Concurrent Monitor** memory. The Concurrent Monitor operates in sequential fixed block mode which means that the 1553 Message blocks are stored in sequential locations in memory, beginning from block 0.

The first call to the function Get_Next_Message_BCM_Px returns the contents of block 0. If the next block does not contain a new message, an error value will be returned.

Note: The structure returned by this function is similar to that returned by the Sequential Monitor mode function Get_Next_Message_Px. However, the bits used in the Message Status word of the **Internal Concurrent Monitor** are unique to this function – they are similar but not identical to the bits set in the Message Status word of Sequential Monitor mode and BC/Concurrent-RT mode and its Internal Concurrent Monitor.

For each mode verify the Message Status bits for the appropriate mode and function.

Syntax

Get_Next_Message_BCM_Px (`int` handle, `struct MONMSG` *msgptr, `usint` *param3)

Input Parameters

handle — The handle designated by Init_Module_Px

Output Parameters

msgptr — Pointer to the address in dual-port RAM at which the message is found

struct MONMSG {

int msgstatus
Status Word containing one or more of the following flags:

END_OF_MSG	Indicates end of message [8000 H]
BUS_A	Indicates BUS A [4000 H]
BAD_CHECKSUM_CONCM	Checksum error [2000 H] (for 1760 only)
ME_SET	Message Error bit in the RT Status word [1000 H]
BAD_STATUS	Status Word bits set [0800 H]
INVALID_MSG	Invalid message; perhaps RT-to-RT with two receive Command words [0400 H]

Get_Next_Message_BCM_Px (cont.)

	MON_LATE_RESP	Response time error occurred in the message, even if no RT active on the module [0200 H]
	BAD_HEADER_BCCM	Bad Header Word [0100 H] (for 1760 only)
	INVALID_WORD	At least one invalid 1553 Word received [0080 H]
	WD_CNT_HI	RT transmitted too many words [0040 H]
	WD_CNT_LO	RT transmitted too few words [0020 H]
	BAD_RT_ADDR	Received 1553 Status word did not contain the correct RT address [0010 H]
	BAD_SYNC	Sync of either the Command or the Data word(s) is incorrect [0008 H]
	BAD_GAP	Invalid gap received between 1553 Words [0004 H]
	RT2RT_MSG_CONCM	RT-to-RT message was received [0002 H]
	MSG_ERROR	Error occurred; defined in other flags [0001 H]
unsigned long elapsedtime		Always a 32-bit-Time Tag; resolution is 4 microseconds
unsigned int words [36];		A pointer to an array of 1553 words in the sequence they were received over the bus.
}		
param3		Reserved for future use

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
enomsg	If there is no message to return
block number	If successful. Valid values are 0 – 408

Note: For *M4K1553Px* modules rev A and rev A1 using firmware 1.6 or earlier: valid values are 0 – 127.

Insert_Msg_Err_Px

Description

Insert_Msg_Err_Px selects various errors to inject into a previously defined message. This functions may also be called in realtime after a call to Run_BC_Px.

Syntax

Insert_Msg_Err_Px (`int` handle, `int` id, `int` flags)

Input Parameters

handle	The handle designated by Init_Module_Px	
id	Message identifier returned from a prior call to Create_1553_Message_Px	
flags	PARITY_ERR	Select parity error on 1553 words [4000 H]
	WD_CNT_ERR	Inserts bad word count based on Set_Word_Cnt_Px [1000 H]
	BIT_CNT_ERR	Inserts bad bit count based on Set_Bit_Cnt_Px [0800 H]
	SYNC_ERR	Data type sync is inserted in Command word [0400 H]
	GAP_ERR	Invalid gap time between Command and Data words [0300 H]
	DATA_ERR	For BC-to-RT commands the SYNC_ERR and PARITY_ERR may be accompanied by this flag to cause the error to be inserted in the Data words rather than the Command word [0100 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If id is not a valid message id
0	If successful

Re_Create_Message_Px

Description

Re_Create_Message_Px enables the user to alter all the parameters of a message for a particular message id that has already been allocated. Re_Create_Message_Px has the same parameters as Create_1553_Message_Px except that the id is an input instead of an output.

To change a message, the user must give the message's message id. The total message size must not exceed the message size of the one allocated for the original message that was created by Create_1553_Message_Px.

Note: See **Appendix A: MIL-STD-1553 Formats**

Syntax

Re_Create_Message_Px (`int` handle, `usint` cmdtype, `usint` data[], `short int` id)

Input Parameters

handle	The handle designated by Init_Module_Px	
cmdtype	One of the following flags:	
	RT2BC	Send a transmit message [0000 H]
	BC2RT	Send a Receive message [0001 H]
	RT2RT	Send an RT-to-RT transfer message [0002 H]
	MODE	Mode Command [0003 H]
	BRD_RCV	Broadcast Receive message [0004 H]
	BRD_RT2RT	Broadcast RT-to-RT transfer message [0005 H]
	BRD_MODE	Broadcast Mode Command [0006 H]
	MINOR_FRAME	Minor frame message - delay [000F H]
data []	A pointer to an array of command + data (and Status words if RT is simulated by this module)	
id	Message identifier returned by Create_1553_Message_Px	

Output Parameters none

Re_Create_Message_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	ebadid	If id is not a valid message id
	einval	If parameter is out of range
	ebadcommandword	If the T/R bit in the Command word is set incorrectly for the given cmdtype parameter
	msg2big	If attempted to create a message with too many words
	0	If successful

Read_Message_Px

Description Read_Message_Px allows the user to read back data associated with a given message. This is used for reading data returned from a Transmit command or for reading the 1553 Status word returned by an RT for all message types. The output consists of the entire message in the same sequence as it appeared on the bus including Command words, Status words and data.

Note: See **Appendix A: MIL-STD-1553 Formats**

Syntax Read_Message_Px (`int` handle, `int` id, `usint` *words)

Input Parameters	handle	The handle designated by Init_Module_Px
	id	Message identifier returned from a prior call to Create_1553_Message_Px
Output Parameters	words	An array of up to 36 words containing Command, Status and Data words associated with the message id
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	ebadid	If id is not a valid message id
	eminorframe	If message is a Minor Frame type
	0	If successful

Read_SRQ_Message_Px

Description Read_SRQ_Message_Px allows the user to read back the Command and Data Words associated with the last SRQ message

Syntax Read_SRQ_Message_Px (`int` handle, `usint` *vector_status, `usint` *msg_status, `usint` *data, `usint` *srq_counter)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

vector_status	Status word of the Mode code Get Vector Command
msg_status	Status of the BC-to-RT message
Note:	For msg_status/vector_status flag values, see Get_Msgentry_Status_Px on page 5-20.
data	Pointer to an array of up to 36 words associated with the SRQ message
srq_counter	The number of times an SRQ message came in

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
enosrq	If SRQ was disabled by the user See Enable_SRQ_Support_Px on page 5-16
0	If successful

Reset_BC_Status_Px

Description Reset_BC_Status_Px clears the BC status so that the next call to Get_BC_Status_Px is meaningful.

Syntax Reset_BC_Status_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
0	If successful

Run_BC_Px

Description Run_BC_Px causes the Bus Controller to start transmitting messages. All data structures must be set up before calling this function.

Syntax Run_BC_Px (`int` handle, `int` type)

handle	The handle designated by Init_Module_Px	
type	0	Continuous operation
	1 – 255	The number of times to execute the current frame

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
bcr_erange	If called with type greater than 255
etimeout	If module is currently running; module was not in halted state
0	If successful

Select_Async_Frame_Px

Description With multiple calls to Create_Frame_Px the user can create multiple frames. Select_Async_Frame_Px selects a frame to be set up for asynchronous transmission.

Note: Always call Select_Async_Frame_Px *before* Send_Async_Frame_Px.

Syntax Select_Async_Frame_Px (`int` handle, `int` frameid, `int` nummsgs)

Input Parameters

handle	The handle designated by Init_Module_Px	
frameid	Frame identifier returned from a prior call to Create_Frame_Px	
nummsgs	Number of messages within the frame to execute	
	or	
	FULLFRAME	Execute the entire frame [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
frm_erange	If called this function with an undefined frame
frm_erangecnt	If called to this function with a nummsgs value greater than the number of messages in the frame
0	If successful

Send_Async_Frame_Px

Description Send_Async_Frame_Px causes the asynchronous frame that was set up by a call to Select_Async_Frame_Px to be sent after the current message has completed its transmission.

Note: Always call Select_Async_Frame_Px *before* Send_Async_Frame_Px.

Syntax Send_Async_Frame_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
enoasync	If there are not enough messages in async frame
0	If successful

Send_Timetag_Sync_Px

Description Send_Timetag_Sync_Px sets the BC to send the current Time tag value as the Data word in a Mode Code Synchronize with Data message (Mode code 17). The Time tag is sent out as a single 16-bit Data word, with a resolution of 64 μsec.

Note: When this feature is disabled, the BC sends the Data word as defined in the message.

Syntax Send_Timetag_Sync_Px (`int` handle, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px	
flag	ENABLE	Send Time tag as data for Mode code 17 [0001 H]
	DISABLE	Do *not* send Time tag as data for Mode code 17 [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
einval	If parameter is out of range
0	If successful

Set_BC_Resp_Px

Description Set_BC_Resp_Px sets the amount of time the BC will wait for a response before declaring the RT late.

Syntax Set_BC_Resp_Px (`int` handle, `int` time)

handle	The handle designated by Init_Module_Px
time	Time in nanoseconds Valid range: 2000 – 32000

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
einval	If an invalid parameter was used as an input
0	If successful

Set_Bit_Cnt_Px

Description Set_Bit_Cnt_Px determines how many bits will be sent for messages that have requested a BIT_CNT_ERR error injection. The number sent will be the usual number of bits in a word (20) + offset.

Syntax Set_Bit_Cnt_Px (`int` handle, `int` offset)

Input Parameters

handle	The handle designated by Init_Module_Px
offset	Number of bits to add or subtract from the correct number of bits in a 1553 word (20). Valid values:-3 to +3

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
einval	If an invalid parameter was used as an input
0	If successful

Set_Bus_Px

Description Set_Bus_Px sets the bus over which a particular message will be transmitted.

Syntax Set_Bus_Px (`int` handle, `int` id, `int` bus)

Input Parameters

handle	The handle designated by Init_Module_Px	
id	Message identifier returned from a prior call to Create_1553_Message_Px	
bus	XMT_BUS_A	Transmit message over Bus A [0080 H]
	XMT_BUS_B	Transmit message over Bus B [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If id is not a valid message id
ebadchan	If tried to set bus to illegal value
0	If successful

Set_Continue_Px

Description Set_Continue_Px restarts transmission that has been halted after a call to Set_Halt_Px.

Syntax Set_Continue_Px (`int` handle, `int` id)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If id is not a valid message id
0	If successful

Set_Error_Location_Px

Description Set_Error_Location_Px sets the location of errors injected into messages.

Syntax Set_Error_Location_Px (`int` handle, `unsigned int` errorWord, `unsigned int` zcerrorBit)

Input Parameters

handle	The handle returned by Init_Module_Px
errorWord	The index of the data word into which to inject the error. This is used for data word errors only. 0 represents the first data word, 1 the second, etc.
zcerrorBit	The index of the bit into which to inject a zero crossing error. 0 represents the high bit, (i.e. the first bit to be transmitted after the sync) 1 the next, etc.

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
func_invalid	If the function is not supported by this version of the firmware (1.10 and up of the *M4K1553Px* or the *EXC-1553PCMCIA/EPII*)
emode	If the module is not in BC/Concurrent RT mode
einval	If errorWord is greater than 31 or zcerrorBit is greater than 15
0	If successful

Set_Frame_Time_Px

Description Set_Frame_Time_Px determines the time each frame will take when the frame is to be executed more than once. (See Run_BC_Px on page 5-28). This value is the minimum time in microseconds from the start of the frame to the start of the next repetition of the frame.

Syntax Set_Frame_Time_Px (`int` handle, `long` time)

Input Parameters

handle	The handle designated by Init_Module_Px
time	Frame time in microseconds

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
0	If successful

Set_Halt_Px

Description Set_Halt_Px halts transmission upon reaching this message. Transmission will begin again after Set_Continue_Px is called.

Syntax Set_Halt_Px (`int` handle, `int` id)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If id is not a valid message id
0	If successful

Set_Interrupt_On_Msg_Px

Description Set_Interrupt_On_Msg_Px causes an interrupt to be generated when a chosen message is generated.

Syntax Set_Interrupt_On_Msg_Px (`int` handle, `int` frameid, `int` msgentry, `int` enable)

Input Parameters

handle	The handle designated by Init_Module_Px	
frameid	frame_id returned from a call to Create_Frame_Px	
msgentry	Message number within the frame	
enable	ENABLE	Generate an interrupt [0001 H]
	DISABLE	Do *not* generate an interrupt [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
frm_erange	If frameid is not a valid frame id
frm_erangecnt	If msgentry is greater than the number of messages in the frame
emode	If not in BC/Concurrent-RT mode
0	If successful

Set_Jump_Px

Description Set_Jump_Px sets the module to jump to a new frame when the module gets to the selected message.

Note: The original message denoted by the id is destroyed by this function.

Syntax Set_Jump_Px (`int` handle, `int` id, `int` frameid, `int` intcnt)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px for jumping message
frameid	Frame identifier returned from a prior call to Create_Frame_Px for frame to jump to
intcnt	Number of instructions in new frame to execute *or* 0 Execute all instructions in frame

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If id is not a valid message id
frm_erange	If frameid is not a valid frame id
frm_erangecnt	If intcnt is greater than the number of messages in the frame
0	If successful

Set_Minor_Frame_Time_Px

Description Set_Minor_Frame_Time_Px sets time for a Minor Frame message type. This message functions as a "delay time" message. The time given is from the beginning of a Minor Frame to the beginning of the next Minor Frame.

Syntax Set_Minor_Frame_Time_Px (`int` handle, `long` micro_second)

Input Parameters

handle	The handle designated by Init_Module_Px
micro_second	Minor Frame time in microseconds; maximum value allowed is 800,000 microseconds (800 milliseconds)

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
0	If successful

Set_Replay_Px

Description Set_Replay_Px sets the module in replay mode. When replay mode is enabled, a message will be sent out when the Time tag equals the message send time (mst) as specified in that msgentry within the frame.

Syntax Set_Replay_Px (`int` handle, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px	
flag	ENABLE	Enable replay mode [0001 H]
	DISABLE	Disable replay mode [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
einval	If a parameter is out of range
0	If successful

Set_Restore_Px

Description Set_Restore_Px restores a message, that was set to skip, to be sent as usual.

Syntax Set_Restore_Px (`int` handle, `int` id, `int` msgtype)

Input Parameters

handle	The handle designated by Init_Module_Px	
id	Message identifier returned from a prior call to Create_1553_Message_Px	
msgtype	One of the following:	
	RT2BC	Send a transmit message [0000 H]
	BC2RT	Send a Receive message [0001 H]
	RT2RT	Send an RT to RT transfer message [0002 H]
	MODE	Mode Command [0003 H]
	BRD_RCV	Broadcast Receive message [0004 H]
	BRD_RT2RT	Broadcast RT to RT transfer message [0005 H]
	BRD_MODE	Broadcast Mode Command [0006 H]
	MINOR_FRAME	Minor frame message - delay [000F H]

Note: The msgtype must be the same as was used in Create_1553_Message_Px. Set_Skip_Px destroys this information, therefore Set_Restore_Px must supply it again.

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If id is not a valid message id
einval	If an invalid parameter was used as an input
enoskip	If tried to restore a message that was not skipped
0	If successful

Set_Retry_Px

Description Set_Retry_Px selects the number and type of retries to attempt in the event of an error.

Syntax Set_Retry_Px (`int` handle, `int` id, `int` num, `int` altbus)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px
num	Number of retries to attempt: 1 – 3 Number of valid retries 0 No retries
altbus	Whether to retry on the initial bus or to alternate buses for each retry. RETRY_SAME_BUS Retries should all be on the original bus [0000 H] RETRY_ALT_BUS Retries should alternate between buses [0040 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If id is not a valid message id
einval	If an invalid parameter was used as an input
0	If successful

Set_RT_Active_Px

Description Set_RT_Active_Px causes a particular RT to be simulated.

Note: This function is for use in BC/Concurrent-RT mode. Ignore this function if only a Bus Controller is simulated.

Syntax Set_RT_Active_Px (`int` handle, `int` rtnum, `int` intrpt)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	Address of the RT Valid values: 0 – 31
intrpt	This parameter is not used in BC/ Concurrent-RT mode. It is used in RT mode.

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module not in BC/Concurrent-RT mode
einval	If an invalid parameter was used as an input
0	If successful

Set_RT_Nonactive_Px

Description Set_RT_Nonactive_Px sets a particular RT address nonactive, thereby turning off simulation for the specified RT.This function may be called in BC/Concurrent-RT mode to turn off Concurrent-RT simulation.

Syntax Set_RT_Nonactive_Px (`int` handle, `int` rtnum)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	Address of the RT Valid values: 0 – 31

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module not in BC/Concurrent-RT mode
0	If successful

Set_RT_Resp_Time_Px

Description
Set_RT_Resp_Time_Px sets response time for RTs. This is the time between receipt of the command (and data for BC-to-RT messages) and the transmission of the 1553 Status word by the RT.
This function may be called in BC/Concurrent-RT mode to adjust the timing of simulated Concurrent-RTs.

Syntax
Set_RT_Resp_Time_Px (`int` handle, `int` nsecs)

Input Parameters

handle	The handle designated by Init_Module_Px
nsecs	Time in nanoseconds Valid range: 4000 – 42000

Output Parameters
none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module not in BC/Concurrent-RT mode
0	If successful

Set_Skip_Px

Description
Set_Skip_Px causes a message within a specified frame to be skipped. The message can be restored later by calling Set_Restore_Px.

Syntax
Set_Skip_Px (`int` handle, `int` id)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px

Output Parameters
none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If id is not a valid message id
0	If successful

Set_Stop_On_Error_Px

Description Set_Stop_On_Error_Px sets the module to halt transmission upon reaching a message for which this function has been called and which returns an error status after all retry attempts.

Syntax Set_Stop_On_Error_Px (`int` handle, `int` id)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module not in BC/Concurrent-RT mode
ebadid	If id is not a valid message id
0	If successful

Set_Sync_Pattern_Px

Description Set_Sync_Pattern_Px sets the sync pattern to be used when a Sync error is injected into a message.

Syntax Set_Sync_Pattern_Px (`int` handle, `int` pattern)

Input Parameters

handle	The handle returned by Init_Module_Px
pattern	Six bits (in the low six bits of this register) each representing a half bit time of the desired register. For example, a value 0023 H (100011 binary) would be sent as one bit times high, three bit times low and two bit times high.

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
func_invalid	If the function is not supported by this version of the firmware (requires firmware version 1.10 and up of the *M4K1553Px* or the *EXC-1553PCMCIA/EPII*)
emode	If module is not in BC/Concurrent-RT mode
einval	If pattern contains a set bit above the low six bits
0	If successful

Set_Var_Amp_Px

Description	The Set_Var_Amp_Px function enables the user to set the amplitude of the 1553 output signal.	
Syntax	Set_Var_Amp_Px (`int` handle, `int` millivolts)	
Input Parameters	handle	The handle designated by Init_Module_Px
	millivolts	Amplitude in millivolts. Value will be rounded up to the nearest 30 millivolts. Valid range: 0 – 7.5 volts.
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	einvamp	If tried to set invalid amplitude
	0	If successful

Set_Word_Cnt_Px

Description	Set_Word_Cnt_Px sets the number of words to send for messages that have requested WD_CNT_ERR errors. The number sent will be the actual number of words in the message + offset.	
Syntax	Set_Word_Cnt_Px (`int` handle, `int` offset)	
Input Parameters	handle	The handle designated by Init_Module_Px
	offset	Offset from correct word count Valid values: -3 to +3
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module not in BC/Concurrent-RT mode
	einval	If an invalid parameter was used as an input
	0	If successful

Set_Zero_Cross_Px

Description Set_Zero_Cross_Px sets the type of zero crossing to be used when a Zero Crossing error is injected into a message.

Syntax Set_Zero_Cross_Px (`int` handle, `unsigned int` zctype)

Input parameters

handle	The handle returned by Init_Module_Px	
zctype	ZC_NO_ERROR	No error [0000 H]
	ZC_LATE100	Zero cross 100 nanoseconds late (legal) [0001 H]
	ZC_LATE150	Zero cross 150 nanoseconds late (illegal) [0002 H]
	ZC_LATE200	Zero cross 200 nanoseconds late (illegal) [0003 H]
	ZC_EARLY100	Zero cross 100 nanoseconds early (legal) [0004 H]
	ZC_EARLY150	Zero cross 150 nanoseconds early (illegal) [0005 H]
	ZC_EARLY200	Zero cross 200 nanoseconds early (illegal) [0006 H]
	ZC_STUCK_HI	No zero cross, full bit hi [0008 H]
	ZC_STUCK_LO	No zero cross, full bit low [0009 H]
	ZC_STUCK_DEAD	No zero cross, full bit dead (center) [000A H]

Output parameters none

Return values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
func_invalid	If the function is not supported by this version of the firmware (requires firmware version 1.10 and up of the *M4K1553Px* or the *EXC-1553PCMCIA/EPII*)
emode	If the module is not in BC/Concurrent RT mode
einval	If zctype is not one of the flags listed
0	If successful

Start_Frame_Px

Description

Start_Frame_Px selects which of the multiple frames created by calls to Create_Frame_Px are to be run when Run_BC_Px is called.

Note: This function must be called prior to each call on Run_BC_Px. If the module is stopped, either by calling Stop_Px, or because all requested messages have been transmitted, the module may be restarted by calling Start_Frame_Px and then Run_BC_Px

Syntax

Start_Frame_Px (`int` handle, `int` frameid, `int` intcnt)

Input Parameters

handle	The handle designated by Init_Module_Px
frameid	Frame identifier returned from a prior call to Create_Frame_Px
intcnt	Number of messages within the frame to execute *or* FULLFRAME Execute the entire frame [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
frm_erange	If frameid is not a valid frame id
frm_erangecnt	If intcnt is greater than the number of messages in the frame
0	If successful

Appendix A MIL-STD-1553 Formats

Appendix A provides diagrams of MIL-STD-1553 Word and Message formats.

A-1 Word Formats

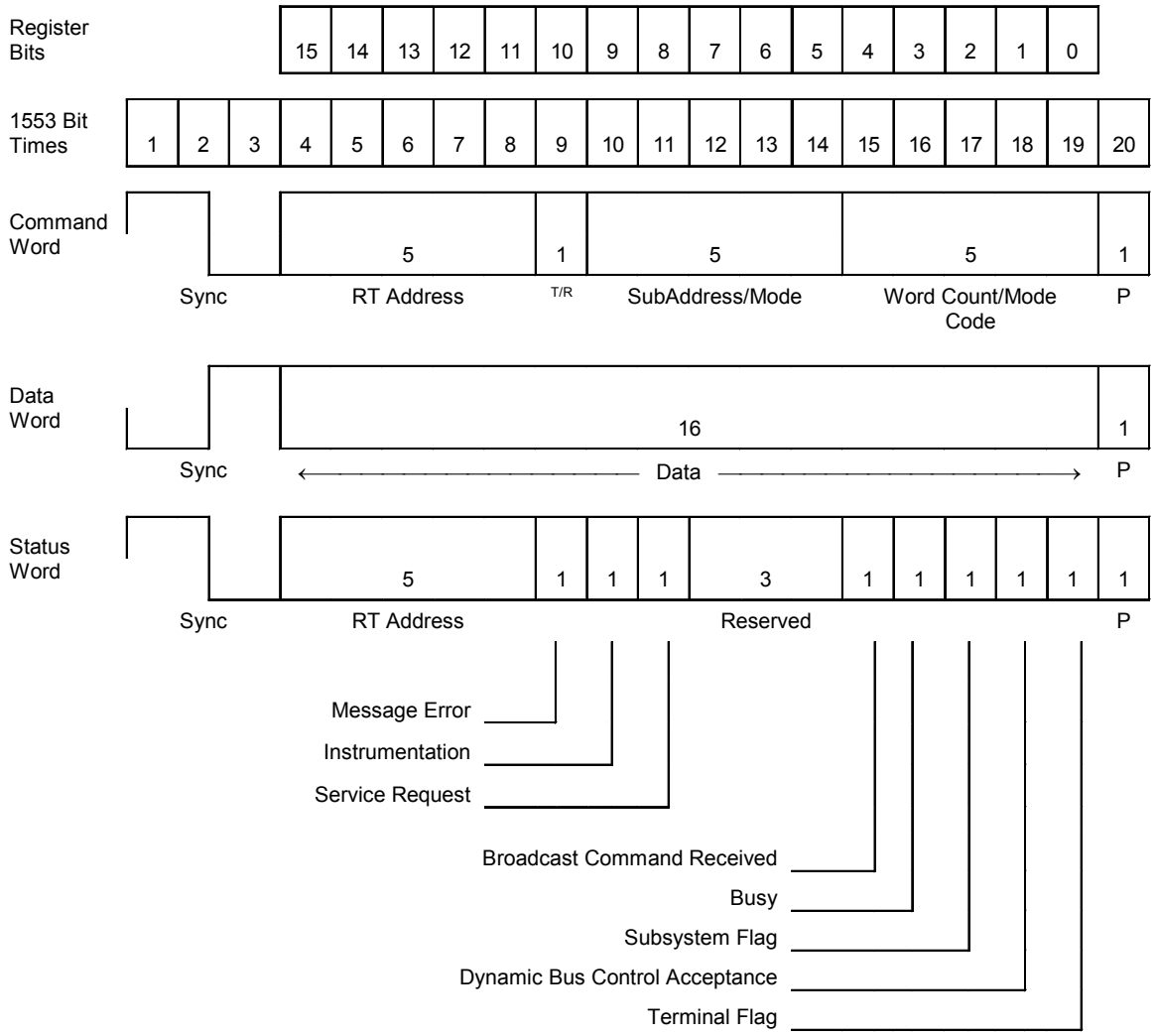


Figure A-1 MIL-STD-1553 Word Formats

Note: T/R = Transmit/Receive
P = Parity

A-2 Message Formats

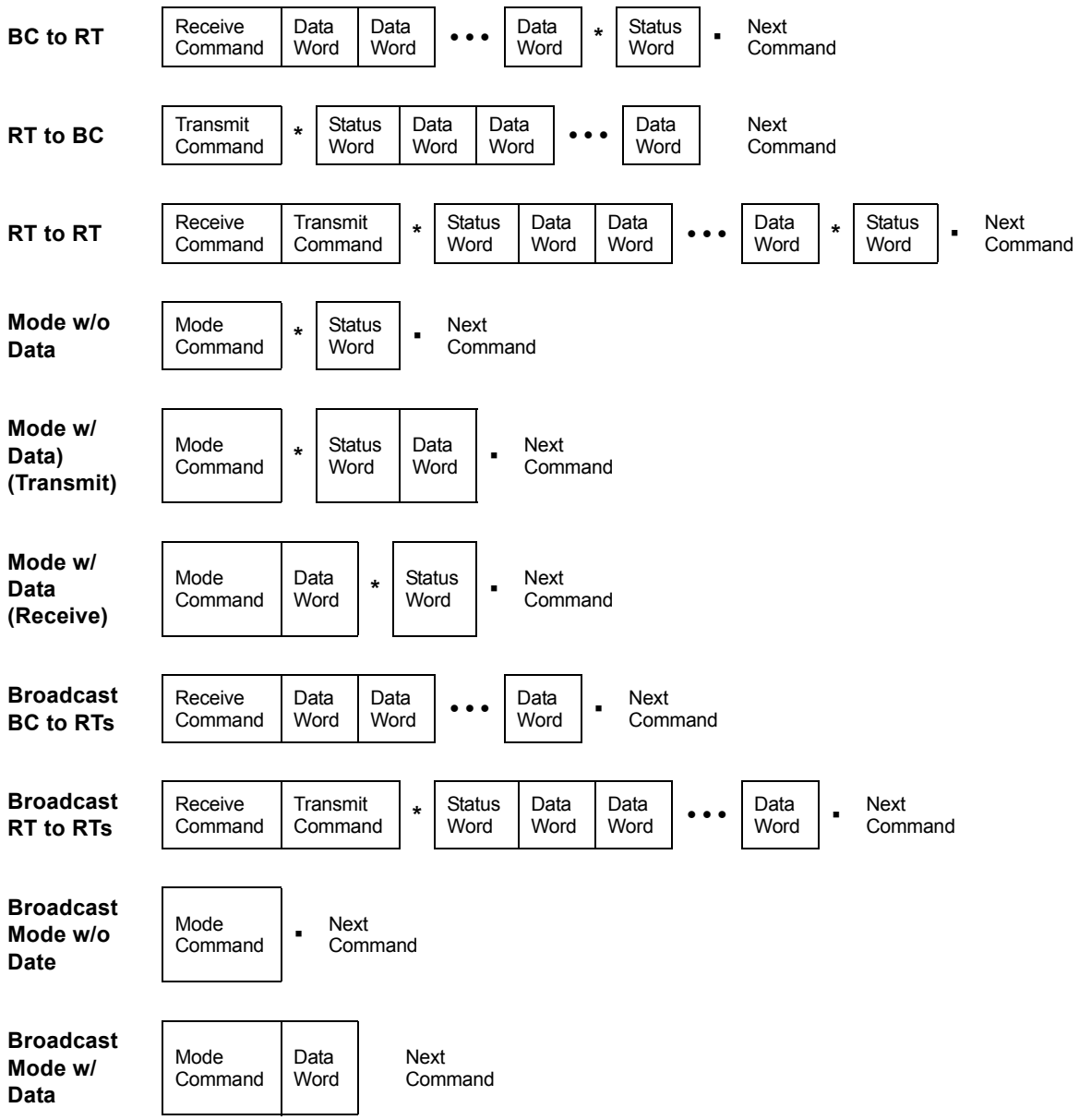


Figure A-2 MIL-STD-1553B Message Formats

Note: * = Response time
▪ = Intermessage Gaptime

Appendix B *Galahad Software Tools* Installation Instructions

Appendix B provides installation instructions for adding*M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools* to the following boards:

Appendix B-1 EXC-4000PCI Boards B-1
Windows 9x /ME Procedures B-1
Windows NT4 Procedures B-3
Windows 2000/XP Procedures B-4
Appendix B-2 EXC-4000VME Boards B-6
Installation Instructions for PCI-MXI-2 Systems B-6
Appendix B-3 PCMCIA Cards B-7
Windows 9*x*/ME Procedures B-7
Windows NT4 Procedures B-9
Windows 2000/XP Procedures B-10

Appendix B-1 EXC-4000PCI Boards

Appendix B-1 explains the procedure for installing an EXC-4000PCI or EXC-4000cPCI board and adding *M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools* to run under the following operating systems:

- Windows 9*x/ME*
- Windows NT4
- Windows 2000/XP

Warning: Whenever you handle an EXC-4000PCI board, wear a suitably grounded electrostatic discharge wrist strap.

Windows 9*x /ME* Procedures

The *Excalibur Installation CD* contains the files for:

- *PCI Hardware Installation for Windows 9x/ME*
- *M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools*

To install an EXC-4000PCI board on a Windows 9*x* system:

1. Insert the board into an available PCI slot in your computer
2. Add *Galahad Software Tools* to your Windows 9*x* system.

To verify that the board was installed correctly, run the test programs included on the *Excalibur Installation CD*.

To install the Excalibur EXC-4000PCI board

1. Make sure the computer is turned off. Insert the board into one of the available slots. For more information see the Installation section in the hardware *User's Manual.*
2. Turn on the computer and wait several moments while Windows 9*x/ME* boots up.
3. The message **Building driver information data** may or may not appear. After several seconds, the message **New Hardware Found** is displayed. Next, the **Update Device Driver Wizard** is displayed.

Note: If the **Update Device Driver Wizard** is not displayed, and you are upgrading from a previous version of the PCI Win 9*x*/ME Hardware Installation, follow these steps to invoke the **Update Device Driver Wizard.** Right-click **My Computer,** then click **Properties | Device Manager | Excalibur PCI Cards | Excalibur 4000PCI Card | Properties | Driver | Update Driver.**

4. Insert the *Excalibur Installation CD* in the drive and follow the on-screen instructions.
5. When the following message appears: **Windows found the following updated driver for this device: Excalibur EXC-4000 card**, click **Finish**.
6. Board installation is completed. If prompted to reboot the computer, do so now. Next, continue with board verification below.

To verify board installation:

1. Make certain the Excalibur EXC-4000PCI board is in place in the computer.
2. Right-click **My Computer**. Select **Properties.** The **System Properties** dialog box appears.
3. In the **System Properties** dialog box, click the **Device Manager** tab.
4. Double-click **Excalibur PCI Card**. Verify that the Excalibur EXC-4000PCI board is listed next to a gray diamond-shaped icon.

Board installation verification is successful.

Note: If you see an exclamation point (**!**) superimposed on the gray diamond, this indicates that the board is not properly installed. Check the following:

- There are not enough memory resources available in the system, free up more memory or IRQs.
- That the computer is Plug and Play compatible. Many computers claim to be "Plug and Play" but are not fully compatible with the Plug and Play specification.

To Add *Galahad Software Tools* to Windows 9*x/ME* Systems

Note: If there is already a previous version of *Galahad Software Tools* for the same module installed on the computer, the new version will overwrite it. We recommend that you first uninstall the previous version by selecting **Start | Settings | Control Panel | Add/Remove Programs.** If you want to save the earlier version, you can choose to install the new version to a different directory when the Install Wizard asks to which directory to install the software.

1. Insert the *Excalibur Installation CD* in the drive, click **Install Drivers, Applications, Utilities**.
2. Follow the on-screen instructions to select the software that matches your product.
3. When the **Excalibur Configuration Utility** (**`ExcConfig`**) screen appears, double-click the **Type** field. Select **4000PCI**.
4. Leave the **Auto** values and click **Save**.

Note: Remember the device number; it is the parameter for the Init_Module_Px function.

The EXC-4000PCI board is now ready to run.

Running test programs

Excalibur provides test programs to verify that the board is operating properly. The source code is provided with the test programs as a guide to develop your own applications.

To run the test programs in *Galahad Software Tools,* the EXC-4000PCI must be linked up to a loopback cable. For details refer to the section *Mechanical and Electrical Specification* in your specific module *User's Manual.*

Go to **Start | Programs | *[Product Name]***, to run the test programs.

Windows NT4 Procedures

The *Excalibur Installation CD* contains the files for *Galahad Software Tools*.

To install an EXC-4000PCI board on a Windows NT4 system:

1. Insert the board into an available PCI slot in the computer.
2. Add *Galahad Software Tools*
3. Reboot the computer
4. To verify that the board was installed correctly, run the test programs that are included on the *Excalibur Installation CD*

To install the Excalibur EXC-4000PCI board

1. Make sure the computer is turned off.
2. Insert the board into one of the available slots.

For more information, see the section **Installation** in the hardware module *User's Manual.*

To add *Galahad Software Tools* to Windows NT4 systems

Note: If there is already a previous version of *Software Tools* for the same module installed on the computer, the new version will overwrite it. We recommend that you first uninstall the previous version by selecting **Start | Settings | Control Panel | Add/Remove Programs**. If you want to save the earlier version, you can choose to install the new version to a different directory when the Install Wizard asks to which directory to install the software.

1. Insert the *Excalibur Installation CD* in the drive, click **Install Drivers, Applications, Utilities**.
2. Follow the on-screen instructions to select the software that matches your product.
3. When the **Excalibur Configuration Utility** (**`ExcConfig`**) screen appears, double-click the Type field. Select **4000PCI**.
4. Leave the **Auto** values and click **Save**.

   Note: Remember the device number; it is the parameter for the Init_Module_Px function.
5. Reboot the computer at the end of the installation procedure.

The EXC-4000PCI board is now ready to run.

Running test programs

Excalibur provides test programs to verify that the board is operating properly. The source code is provided with the test programs as a guide to develop your own applications.

To run the test programs in *Galahad Software Tools,* the EXC-4000PCI must be linked up to a loopback cable. For details refer to the section *Mechanical and Electrical Specification* in your specific module's *User's Manual.*

Go to **Start | Programs | *[Product Name]***, to run the test programs.

Windows 2000/XP Procedures

The *Excalibur Installation CD* contains the files for:

- *PCI Hardware Installation for Windows 2000/XP*
- *M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools*

To install an EXC-4000PCI board on a Windows 2000/XP system:

1. Insert the board into an available PCI slot in your computer
2. Add *Galahad Software Tools* to your Windows 2000/XP system.

To verify that the board was installed correctly, run the test programs included on the *Excalibur Installation CD*.

To install the Excalibur EXC-4000PCI Board:

1. Make sure the computer is turned off. Insert the board into one of the available slots. For more information see the Installation section in the module's *User's Manual.*
2. Turn on the computer and wait several moments while Windows 2000/XP boots up.
3. The message **Building driver information data** may or may not appear. After several seconds, the message **New Hardware Found** is displayed. Next, the **Update Device Driver Wizard** is displayed.

Note: If the **Update Device Driver Wizard** is not displayed, and you are upgrading from a previous version of the PCI Windows Hardware Installation, follow these steps to invoke the **Update Device Driver Wizard.** Right-click **My Computer,** then click **Hardware | Device Manager | Excalibur PCI Cards | Excalibur 4000PCI card.** Double-click and select **Driver | Update Drivers**.

4. Insert the *Excalibur Installation CD* in the drive and follow the on-screen instructions.
5. When the following message appears: **Windows found the following updated driver for this device: Excalibur EXC-4000 card**, click **Finish**.
6. Board installation is completed. If prompted to reboot the computer, do so now. Next, continue with board verification below.

To verify board installation:

1. Make certain the Excalibur EXC-4000PCI board is in place in the computer.
2. Right-click **My Computer | Properties**. The System Properties dialog box appears.
3. In the System Properties dialog box, click **Hardware | Device Manager.**
4. Double-click **Excalibur PCI Board**. Verify that the Excalibur PCI Board is listed next to a gray diamond-shaped icon.

Note: If an exclamation point **(!)** appears, superimposed on the gray diamond, this indicates that the board is not properly installed. Check that there are enough memory resources available in the system. Free up more memory or IRQs.

The board installation verification is successful.

To add *Galahad Software Tools* under Windows 2000/XP

Note: If there is already a previous version of *Software Tools* for the same module installed on the computer, the new version will overwrite it. We recommend that you first uninstall the previous version by selecting **Start | Settings | Control Panel | Add/Remove Programs**. If you want to save the earlier version, you can choose to install the new version to a different directory when the Install Wizard asks to which directory to install the software.

1. Insert the Excalibur Installation CD in the drive, click **Install Drivers, Applications, Utilities.** Follow the on-screen instructions to select the software that matches your product.
2. When the **Excalibur Configuration Utility** (**`ExcConfig`**) screen appears, double-click the **Type** field. Select **4000PCI**.
3. Leave the **Auto** values and click **Save**.

Note: Remember the device number; it is the parameter for the Init_Module_Px function.

The EXC-4000PCI board is now ready to run.

Running test programs

Excalibur provides test programs to verify that the board is operating properly. The source code is provided with the test programs as a guide to develop your own applications.

To run the test programs in *Galahad Software Tools,* the EXC-4000PCI must be linked up to a loopback cable. For details refer to the section *Mechanical and Electrical Specification* in your specific module's *User's Manual.*

Go to **Start | Programs | *[Product Name*]**, to run the test programs.

Appendix B-2 EXC-4000VME Boards

Installation Instructions for PCI-MXI-2 Systems

Appendix B-2 explains the procedure for installing the *Galahad Software Tools* for the Excalibur EXC-4000VME and EXC-4000VXI boards under a Windows operating system.

Installation of the EXC-4000VME board on a Windows system is accomplished by determining the board's logical address, setting the DIP switch on the board, and installing *Galahad Software Tools*. To verify board installation, run the test-programs that are included on the *Excalibur Installation CD*.

All the files needed to run *Galahad Software Tools* on Windows operating systems are on the *Excalibur Installation CD*.

Note: *Galahad Software Tools* for VME boards were written for VISA standard.

To install the EXC-4000VME board:

1. Ensure the National Instruments PCI-MXI-2 system is installed correctly.
2. Before installing the Excalibur board, it is very important to determine the board's logical address. The board requires a 1MB area of memory. Choose a logical address that does not conflict with any other devices in your system. The board is set at default address 80H (128 Dec.).

   Warning: If a logical address is already in use and it is also used for the EXC-4000VME, the board will not function properly.
3. Set the Board Logical Address DIP switch (SW1) according to the logical address chosen. See the section on DIP Switch Settings in the EXC-4000VME hardware *User's Manual*.
4. Turn on your VME and PC computers. Run the two programs that establish the connection to the VME (VXIINIT, the hardware initialization program, and Resman, the resources manager program).

Note: Later versions of the National Instruments PCI-MXI-2 may not require running VXIINIT.

Appendix B-3 PCMCIA/EPII Cards

Appendix B-3 explains the procedure for installing *Galahad Software Tools* for the *EXC-1553PCMCIA/EPII* card.

Galahad Software Tools is for several different operating systems.

- Windows 9*x*/ME
- Windows NT4
- Windows 2000/XP

Warning: Whenever you handle an Excalibur card, wear a suitably grounded electrostatic discharge wriststrap.

Inserting a PCMCIA/EPII card:

Insert the Excalibur *PCMCIA/EPII* card. When the card is almost all the way in the slot, push firmly, but gently, to ensure a firm connection with the computer. Do not force the card into position.

Note: The adapter cables may be connected to the card either before or after the card is inserted into the PCMCIA slot on the computer. In addition, the cables may be connected to and disconnected from the card while power to the computer is turned on, but not while the card is transmitting over the bus.

Windows 9*x*/ME Procedures

To install the PMCIA card on a Windows 9*x*/ME system:

1. Turn on the computer.
2. Install the PCMCIA card in your computer
3. Verify the card installation
4. Add *Galahad Software Tools* to your hard disk.
5. Run the test programs included in *Galahad Software Tools*.

The *Excalibur Installation CD* contains the files for:

- Hardware Installation for PCMCIA cards for Windows 9*x*/ME
- *Galahad Software Tools*

To install the PMCIA card:

1. Turn on the computer and wait several moments while Windows 9*x*/ME boots up.
2. Insert the PCMCIA card into any PCMCIA slot. When the card is almost all the way in the slot, push firmly but gently, to ensure a firm connection with the computer. Do not force the card into position. The message **Building driver information data** may or may not appear. After several seconds, the message **New Hardware Found** is displayed. Next, the **Update Device Driver Wizard** is displayed.

Note: If you are upgrading from a previous version of the PCMCIA Hardware Installation and the **Update Device Driver Wizard** is not displayed, right-click **My Computer**, and then click **Properties > Device Manager > EXCALIBUR PCMCIA Cards > your PCMCIA Card > Properties > Drivers > Update Driver**.

3. Insert the *Excalibur Installation CD* in the CD drive and follow the on-screen instructions to select the software that matches your product.
4. When the message: **Windows found the following updated driver for this device: your PCMCIA Card** appears, click **Finish**.
5. Card installation is completed. Reboot the computer.

To verify card installation

1. Make certain the PCMCIA card is in place in the computer.
2. Right-click **My Computer**, select **Properties**. The **System Properties** dialog box appears.
3. In the **System Properties** dialog box, click the **Device Manager** tab.
4. Double-click **Excalibur PCMCIA Cards**. Verify that the PMCIA is listed next to a gray diamond-shaped icon.
5. Card installation verification is successful.

Note: If you see an exclamation point (!) superimposed on the gray diamond, this indicates that the card is not properly installed. Check the following:
- That there are enough memory resources available in your system. If not, free up more memory or IRQs.
- That the computer is Plug and Play compatible. Many computers claim to be "Plug and Play" but are not fully compatible with the Plug and Play specification.

To install *Galahad Software Tools*

Note: If a previous version of this product is installed on the computer, the current installation will overwrite it. We recommend to uninstall the previous version by selecting **Start > Settings > Control Panel > Add/Remove Programs**. To save the previous version, install the new version to a different directory, when the Install Wizard asks to which directory to install the software.

1. Insert *Excalibur Installation CD* in the drive, click **Install Drivers, Applications, Utilities**.
2. Follow the on-screen instructions to select the software that matches your product.
3. When the **Excalibur Configuration Utility** (**`ExcConfig`**) screen appears, double click the Type field. Select [*your PCMCIA card*].

   Note: Remember the device number; it is the parameter for the Init_Module_Px function.
4. Leave the **Auto** values and click **Save**.
5. Reboot the computer at the end of the installation procedure.

Windows NT4 Procedures

The *Excalibur Installation CD* contains the files for *Galahad Software Tools.*

To install *Galahad Software Tools*

Note: If a previous version of this product is installed on the computer, the current installation will overwrite it. We recommend to uninstall the previous version by selecting **Start > Settings > Control Panel > Add/Remove Programs**. To save the previous version, install the new version to a different directory, when the Install Wizard asks to which directory to install the software.

1. Insert *Excalibur Installation CD* in the drive, click **Install Drivers, Applications, Utilities**.
2. Follow the on-screen instructions to select the software that matches your product.
3. When the **Excalibur Configuration Utility** screen appears, double-click the **Type** field, select **PCMCIA**.
   Note: Remember the device number; it is the parameter for the Init_Card_Px function.
4. Enter the appropriate values in Wizard screen, as shown in Table B-1.

Name of Field	Enter the value
Type	See step 2.
Memory base	A 72K area of memory
	To determine which memory areas are available on your computer:
	Click the **START** button on the Windows NT4 Taskbar
	Select **Programs > Administrative Tools > Windows NT Diagnostics > Memory**. The memory areas currently in use are displayed.
	Find a 72K block of unused memory.
	Type the starting address of the unused block in the **Memory base** field.
Interrupts	To determine which interrupts are available on your computer, click the **START** button on the Windows NT Taskbar, then select **Programs > Administrative Tools > Windows NT Diagnostics > Resources**. You will see which Interrupt Numbers are currently in use. Find a number that is not in use (often 5 or 7), and type that number into the Interrupt field.
	Sometimes an Interrupt (IRQ) is not listed as “in use”, but is in fact unavailable. You may need to try several IRQs before you find one that is available and allows you to continue working.
	If you are not using interrupts, it is desirable to leave the Interrupt field undefined.

Table B-1: 1553PCMCIA Memory/IRQ Configuration under Windows NT4

5. Click **OK > Save**.
6. Insert the PCMCIA card and reboot the computer. The card must remain in the computer throughout the reboot operation.

To verify card installation

1. On the Windows NT Taskbar, click **Start > Settings > Control Panel**.
2. Double-click **PCMCIA**. The **PCMCIA Devices screen** appears.
3. On the **PCMCIA Devices screen**, check that **Excalibur Systems Inc.** and PCMCIA is displayed next to a gray diamond-shaped icon. This indicates that your card is properly installed.
4. Click the **Properties** button on the PC Card (PCMCIA) Devices screen, then select the **Card Info** tab. Check that this screen displays the message: **The device is working properly**. This indicates that the card is properly installed.

Procedure for Installing Multiple Cards

Due to the limitations of the Windows NT4 PCMCIA drivers, we cannot support multiple cards on Windows NT4 operating systems.

Running Test Programs

Excalibur provides test programs to verify that the card is operating properly. The source code is provided with these test programs as a guide for you to develop your own applications.

To run a test program, click **Start > Programs > Excalibur > PCMCIA > [*test program icon*]**.

Windows 2000/XP Procedures

The *Excalibur Installation CD* contains the files for:

- Hardware Installation for PCMCIA cards for Windows 2000/XP
- *M4K1553PxII & EXC-1553PCMCIA/EPII Galahad Software Tools*

To Install the Excalibur PCMCIA card

Turn on the computer and wait several minutes while Windows 2000 boots up.

1. Insert the card into one of the available slots. For more information see the Installation section in the *User's Manual* available on the *Excalibur Installation CD*.
2. After several seconds, the **Found New Hardware Wizard: Welcome to the Found New Hardware Wizard** dialog box is displayed. Click **Next**. The **Found New Hardware Wizard: Install Hardware Device Drivers** dialog box is displayed.
3. Insert the *Excalibur Installation CD* in the drive and follow the on-screen instructions to select the software that matches your product.
4. When you get a message **The Wizard found a driver for the following device: *[your PCMCIA]***, click **Next**.
5. The Wizard copies the files on to your hard drive. When the Wizard finishes copying the files **Found New Hardware Wizard: Completing the Found New Hardware Wizard PCMCIA card** is displayed.Windows has finished installing the software for this device. Click **Finish**.

The card installation is complete. If prompted to reboot the computer, do so now.

To verify card installation

1. Make certain the PCMCIA card is in place in the computer.
2. Right-click **My Computer**, select **Properties > Hardware tab > Device Manager.**
3. Double-click Excalibur PCMCIA Cards. Verify that the PCMCIA Card is listed next to a gray diamond-shaped icon.

Card installation verification is successful.

Note: If you see an exclamation point (!) superimposed on the gray diamond, this indicates that the card is not properly installed. Check the following:

- That there are enough memory resources available in your system. If not, free up more memory or IRQs.
- That the computer is Plug and Play compatible. Many computers claim to be "Plug and Play" but are not fully compatible with the Plug and Play specification.

To Install *Galahad Software Tools*

Note: If a previous version of this product is installed on the computer, the current installation will overwrite it. We recommend to uninstall the previous version by selecting **Start > Control Panel > Add/Remove Programs**. To save the previous version, install the new version to a different directory, when the Install Wizard asks to which directory to install the software.

1. Insert *Excalibur Installation CD* in the CD drive, click **Install Drivers, Applications, Utilities**.
2. Follow the on-screen instructions to select the software that matches your product.
3. When the **Excalibur Configuration Utility** (**`ExcConfig`**) screen appears, double-click the **Type** field, choose **[*your PCMCIA*]**.
4. leave the default **Auto** values and click **Save**.

   Note: Remember the device number; it is the parameter for the Init_Module_Px function.

Running Test Programs

Excalibur provides test programs to verify that the card is operating properly. The source code is provided with these test programs as a guide for you to develop your own applications.

To run a test program, click **Start > Programs > Excalibur > PCMCIA > [*test program icon*].**

Appendix C Multiple Board Support

Appendix C-1 EXC-4000PCI and EXC-4000cPCI Boards

Galahad Software Tools supports the use of up to four PCI boards simultaneously.

To use multiple boards:

1. Each board must be set for a unique ID. The ID of a board is set via the SW1 DIP switch. To set the ID of the board set the jumpers as follows:

Board ID	SW1 Switch contact			
	1	2	3	4
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1

Set DIP switch SW1 to a unique ID by setting the switch contacts *open* or *off* to represent logic '1'; and closed or on to represent logic '0'.

2. In the **`ExcConfig`** utility program, create a separate device number for each board. For each board enter the appropriate values in the Excalibur Configuration Screen, as described in the table below.

Name of field	Enter this value
Type	Select 4000PCI from the drop-down list
Unique Identifier	Set to the unique ID configured above in Step 1.
Memory	Set to automatic setting
Interrupt Setup	Set to automatic setting
I/O ports	Set to automatic setting

Appendix C-2 EXC-1553PCMCIA/EPII Cards

Galahad Software Tools supports 2 PCMCIA cards.

1. In the **Excalibur Configuration Utility** program, go to the first empty device. Double-click to display the **Excalibur Configuration Wizard**.
2. In the **Select Card** field, select **[*your PCMCIA*]**.
3. In the **Unique Identifier/Socket Number** field, enter your socket number.

Repeat steps 1-3 for each PCMCIA card installed.

Appendix C-3 EXC-4000VME and EXC-4000VXI Boards

Galahad Software Tools supports the simultaneous use of up to eight modules on a VME or VXI Board. The number of boards supported depends on the users system.

A unique Logical Address must be set for each board. Use DIP switch SW1 to set the board's Logical Address.

Examples of Logical Address Switch (SW1) Settings:

		MSB			**Logical Address Switch (SW1**				LSB
'1'	'1'	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6
Logical Address		**Switch Settings**							
Hex	**Dec**								
1	1	0	0	0	0	0	0	0	1
20	32	0	0	1	0	0	0	0	0
80	128	1	0	0	0	0	0	0	0
81	129	1	0	0	0	0	0	0	1
C0	192	1	1	0	0	0	0	0	0
FF	255	1	1	1	1	1	1	1	1

Example: For a logical address of C0 (H) [=A16 Address F000 (H)], set position 1 and 2 to OFF or Open and all other switches to 0N or Closed.

Switch ON or Closed = logic 0 at switch position
Switch OFF or Open = logic 1 at switch position

Note: 1. Numbers indicate switch positions.
2. Address lines A15 and A14 are always decoded as '1'.
3. Address lines A5 – A0 are always decoded as '0'.

Appendix D Application of External Loopback Test

The external loopback test feature of Excalibur Systems' *EXC-4000* family of boards and the *EXC-1553PCMCIA/EPII* requires a particular Loopback cable configuration. The external Loopback test checks the 1553 transceivers, transformers and associated cables when the board is disconnected from the main MIL-STD-1553 bus. With Bus A and Bus B outputs connected together, the function, **External_Loopback_Px** initiates a special hardware test that transmits Command Sync and Data Sync messages between the two buses.

For proper functioning of the test, a correct stub-to-stub connection must be made. Two methods of properly connecting Bus A and Bus B stubs together are:

- **Direct Coupled Stubs**
- **Transformer Coupled Stubs**

Note: The *EXC-1553PCMCIA/EPII* uses mode-specific connectors. Ignore the references to setting jumpers or DIP switches. For more information contact your Excalibur representative.

Direct Coupled Stubs (Use Direct Coupled Loopback Test Cable, MC1003)

To use direct coupling, set the board's jumpers or DIP switches to direct coupled mode. See the **Mechanical and Electrical** chapter of the *User's Manual* for module's hardware.

For stub lengths of less than one foot, the main MIL-STD-1553 bus can be directly coupled. If this coupling method is being employed, then the Direct Coupled Loopback Test Cable, part number MC1003, must be utilized when running the External Loopback Test. The cable provides one PL75 twinax connector on each end and an integrated 39-ohm termination resistor across the data high and data low lines. To perform the External Loopback Test, disconnect the board's Bus A and Bus B stub connections to the main 1553 bus, connect the test cable to the Bus A and Bus B connections coming from the board and then run the Loopback test. When the External Loopback Test is completed, disconnect the test cable and reconnect the board's Bus A and Bus B stubs to the main MIL-STD-1553 bus.

Transformer Coupled Stubs (Use Universal Test Adapter, ESI-235-1-176-X-XX-XX)

To use transformer coupling, set the board's jumpers or DIP switches to transformer coupled mode. See the **Mechanical and Electrical** chapter of the *User's Manual* for your specific hardware.

This is the preferred method of coupling to the main MIL-STD-1553 bus and requires a bus coupler at the junction of the main bus and stub. If this coupling method is being employed, then the Transformer Coupled Loopback Test Cable, part number ESI-235-1-176-X-XX-XX, must be utilized for running the External Loopback Test. The cable provides one PL75 twinax connector on each end and an integrated two stub internally terminated (39-ohms) in-line bus coupler. To perform the External Loopback Test, disconnect the board's Bus A and Bus B stub connections to the bus couplers, connect the test cable to the Bus A and Bus B connections coming from the board and then run the Loopback test. When the

External Loopback Test is completed, disconnect the test cable and reconnect the board's Bus A and Bus B stubs to the bus couplers.

Note: The stub cables may be connected or disconnected with the board powered on but not while the board is transmitting over the bus.

Appendix E Source Code References

Galahad Software Tools **Library E-2**
Code Index E-5
Error Messages E-9

Appendix E-1 *Galahad Software Tools* Library

Appendix E-1 includes a list of the files in the Excalibur *Galahad Software Tools* needed to write user-defined applications. The files are divided into three categories:

Source code and Header files for the *Galahad Software Tools* functions. Header files should be included in application programs as needed.

File Extension	Description
`*.c`	source code
`*.h`	header file

DLL and associated *.lib files

File Extension	Description
`*.dll`	Borland compiler DLL
`*MS.dll`	Microsoft compiler DLL
`*.lib`	Index file used to create applications using Borland DLL functions
`*MS.lib`	Index file used to create applications using Microsoft DLL functions

Demo Programs are examples of programs using *Galahad Software Tools* which can be used as a basis for user-defined programs. Demo programs include the following types of files:

File Extension	Description
`*.c, *.h`	Demo source code
`*.ide`	Borland demo project files
`*.exe`	Borland demo executable files
`*.dsp` `*.dsw`	Microsoft demo project files
`*MS.exe`	Microsoft demo executable files

	File Name	Description
Source Code files	**bcget**	Functions to retrieve information in BC mode
	bcrtmsg	Functions for BC/Concurrent-RT mode
	bcframe	Functions to set up frame identifiers in BC mode
	bcset	Functions for setting up the module in BC mode
	deviceio_px	Functions for interacting with the Excalibur kernel drivers for all Windows operating systems
	deviceio	Functions relating to resources such as memory and interrupts – these files act on kernel drivers
	error_px	Function for returning error messages
	gget	Functions for retrieving information from all modes
	gset	Functions for setting up the module for non mode-specific values
	initcard	Initialization and Release resources functions
	mon	Functions for Monitor mode
	mon_lkup	Functions for Monitor Lookup mode
	monseq	Functions for Monitor Sequential mode
	rtset	Functions for RT mode
Source Header files	**bcrun**	Header file for BC mode functions
	deviceio	Header file for interaction with kernel driver
	error_px	Header file containing error message codes
	error_devio	Header file containing error codes for the Excalibur module kernel drivers
	excsysio	Header file which defines shared codes and structures between DLL and kernel driver
	exdef	Header file to differentiate between different Excalibur products.
	exc4000	Header file for EXC-4000 board level functions
	galahad	Header file containing all other header files
	flags_px	Header file containing flags for all modes
	instance_px	Header file for global module structure
	monlkup	Header file for Monitor Lookup mode functions
	monseq	Header file for Monitor Sequential mode functions
	proto_px	Header file containing prototypes of all functions
	rtrun	Header file for RT mode functions
	pxIncl	Header file, include file for the Px DLL code, not for applications

	File Name	Description
Demo Programs	**demo_async**	Demo program to test asynchronous functions
	demo_bc1	Demo programs to test basic BC functions
	demo_bc2	Demo program to test advanced BC functions
	demo_2md	Multi-mode demo program to test Message contents and status
	demo_2crd	Demo program runs a BC on one module which sends messages and monitors those messages on the second module. Both modules can be on different EXC-4000 boards or on the same one.
	demo_bcmon	Demo program to test BC/ Internal Concurrent Monitor
	demo_int	Demo program to test interrupts
	demo_mon	Demo program to test Bus Monitor functions
	demo_loopback	Demo program to test External and Internal Loopback functions
	demo_rt	Demo program to test RT functions
	demo_minor	Demo program to test the use of Minor Frames

DLL and *.LIB files

The table below shows the name of the DLL file and its corresponding LIB file for each module under the Borland and Microsoft compilers

Module	Borland	Microsoft
PxII	px	pxms
MCH	mch	mchms
RTx	rtx	rtxms
Discrete	m4kdio	m4kdioms
Serial	excser	excserms
H009	m4h009	m4h009ms
CAN	exccan	exccanms
708	exc708	exc708ms

Appendix E-2 Code Index

Galahad Software Tools is a set of C language functions designed to aid programmers of the P*x* module to write test applications. Below is an alphabetical list of all the functions and the name of *Galahad Software Tools* file containing the programming code. The extension of all the files is *.c

Functions	Code File Name
Alter_Cmd_Px	**BCRTMSG**
Alter_IMG_Px	**BCFRAME**
Alter_Message_Px	**BCRTMSG**
Alter_MsgSendTime_Px	**BCFRAME**
Assign_Blk_Px	**MONLKUP**
Assign_DB_Datablk_Px	**RTSET**
Assign_RT_Data_Px	**RTSET**
BC_Check_Alter_Msgentry_Px	**BCRTMSG**
Clear_Card_Px	**BCSET**
Clear_Frame_Px	**BCSET**
Clear_Msg_Blks_Px	**MON**
Clear_RT_Sync_Entry_Px	**RTSET**
Clear_Timetag_Sync_Px	**BCSET**
Command_Word_Px	**BCRTMSG**
Create_1553_Message_Px	**BCRTMSG**
Create_Frame_Px	**BCFRAME**
Enable_1553A_Support_Px	**GSET**
Enable_Checksum_Error_Px	**BCFRAME**
Enable_Checksum_Px	**BCFRAME**
Enable_Lkup_Int_Px	**MONLKUP**
Enable_SRQ_Support_Px	**BCSET**
External_Loopback_Px	**GGET**
Get_BC_Msgentry_Px	**BCRTMSG**
Get_BC_Msgentry_Px	**BCRTMSG**
Get_BC_Status_Px	**BCGET**
Get_Blknum_Px	**RTSET**
Get_Board_Status_Px	**GGET**
Get_Board_Status_Px	**GGET**
Get_Board_Status_Px	**GGET**
Get_Card_Type_Px	**GGET**
Get_Checksum_Blocks_Px	**BCFRAME**
Get_Counter_Px	**MONSEQ**

Functions	Code File Name
Get_Error_String_Px	**ERROR**
Get_Header_Exists_Px	**GGET**
Get_Header_Value_Px	**GGET**
Get_Id_Px	**GGET**
Get_Interrupt_Count_Px	**DEVICEIO**
Get_Last_Blknum_Px	**MONLKUP**
Get_Message_Px	**MON**
Get_Minor_Frame_Time_Px	**BCGET**
Get_Mode_Px	**GGET**
Get_MON_Status_Px	**MON**
Get_Msgentry_Status_Px	**BCGET**
Get_Next_Message_BCM_Px	**BCRTMSG**
Get_Next_Message_Px	**MONSEQ**
Get_Next_Message_RTM_Px	**RTSET**
Get_Next_RT_Message_Px	**RTSET**
Get_Rev_Level_Px	**GGET**
Get_RT_Message_Px	**RTSET**
Get_RT_Sync_Entry_Px	**RTSET**
Get_Time_Tag_Px	**INITCARD**
Init_Module_Px	**INITCARD**
Init_Module_Px for VME and VXI boards	**INITCARD**
InitializeInterrupt_Px	**DEVICEIO**
Insert_Msg_Err_Px	**BCSET**
Internal_Loopback_Px	**GGET**
Load_Datablk_Px	**RTSET**
Print_Error_Px	**ERROR**
Re_Create_Message_Px	**BCRTMSG**
Read_Datablk_Px	**RTSET**
Read_Message_Px	**BCRTMSG**
Read_RT_Status_Px	**RTSET**
Read_RTid_Px	**RTSET**
Read_SRQ_Message_Px	**BCRTMSG**
Read_Start_Reg_Px	**GGET**
Release_Module_Px	**INITCARD**
Reset_BC_Status_Px	**BCSET**
Reset_RT_Interrupt_Px	**RTSET**
Reset_Time_Tag_Px	**INITCARD**

Functions	Code File Name
Restart_Px	**GSET**
RT_Id_Px	**RTSET**
Run_BC_Px	**BCSET**
Run_MON_Px	**MON**
Run_RT_Px	**RTSET**
Set_1553Status_Px	**RTSET**
Select_Async_Frame_Px	**BCFRAME**
Set_Header_Exists_Px	**GSET**
Set_Header_Value_Px	**GSET**
Send_Async_Frame_Px	**BCFRAME**
Send_Timetag_Sync_Px	**BCSET**
Set_BC_Resp_Px	**BCSET**
Set_Bit_Cnt_Px	**BCSET**
Set_1553Status_Px	**RTSET**
Set_Broad_Ctl_Px	**MON**
Set_Broad_Interrupt_Px	**RTSET**
Set_Bus_Px	**BCSET**
Set_Checksum_Blocks_Px	**BCFRAME**
Set_Cnt_Trig_Px	**MONSEQ**
Set_Continue_Px	**BCSET**
Set_Error_Location_Px	**BCSET**
Set_Frame_Time_Px	**BCSET**
Set_Halt_Px	**BCSET**
Set_Header_Exists_Px	**GSET**
Set_Header_Value_Px	**GSET**
Set_Interrupt_On_Msg_Px	**BCFRAME**
Set_Interrupt_Px	**BCSET**
Set_Invalid_Data_Res_Px	**RTSET**
Set_Jump_Px	**BCSET**
Set_Minor_Frame_Time_Px	**BCSET**
Set_Mode_Addr_Px	**RTSET**
Set_Mode_Px	**GSET**
Set_MON_Concurrent_Px	**MON**
Set_Mon_Response_Time_Px	**MON**
Set_Replay_Px	**BCSET**
Set_Restore_Px	**BCSET**
Set_Retry_Px	**BCSET**

Functions	Code File Name
Set_RT_Active_Bus_Px	**RTSET**
Set_RT_Active_Px	**RTSET**
Set_RT_Broadcast_Px	**INITCARD**
Set_RT_Errors_Px	**RTSET**
Set_RT_Interrupt_Px	**RTSET**
Set_RT_Nonactive_Px	**RTSET**
Set_RT_Resp_Time_Px	**RTSET**
Set_RTid_Interrupt_Px	**RTSET**
Set_RTid_Status_Px	**RTSET**
Set_Skip_Px	**BCSET**
Set_Status_Px	**RTSET**
Set_Stop_On_Error_Px	**BCSET**
Set_Sync_Pattern_Px	**BCSET**
Set_Timetag_Res_Px	**GSET**
Set_Trigger_Mode_Px	**MONSEQ**
Set_Trigger1_Px	**MONSEQ**
Set_Trigger2_Px	**MONSEQ**
Set_Var_Amp_Px	**BCSET**
Set_Vector_Px	**RTSET**
Set_Wd_Cnt_Err_Px	**RTSET**
Set_Word_Cnt_Px	**BCSET**
Set_Zero_Cross_Px	**BCSET**
Start_Frame_Px	**BCFRAME**
Stop_Px	**GSET**
Wait_For_Interrupt_Px	**DEVICEIO**
Wait_For_Multiple_Interrupts_Px	**DEVICEIO**

Appendix E-3 Error Messages

All functions in the *Galahad Software Tools* are written as C functions, i.e they return values. A negative value signifies an error. Below is a list of the error codes, their corresponding negative value and a definition of each error.

Error Code	Value	Definition
ebadid	-1	Undefined message id used as input
einval	-2	Illegal value used as input
emode	-3	Mode specific command called when in the wrong mode
ebadchan	-4	Tried to set channel to illegal value
Init_Card_Mal	-5	Init failed in Simulate mode due to Malloc call failure
msg2big	-6	Attempted to create a message with too many words
msgnospace	-7	Not enough space in message stack for this message
msg2many	-8	Exceeded maximum number of messages permitted (1660)
frm_badid	-9	Attempted to place an undefined message into a message frame
frm_nostacksp	-10	Not enough space in frame stack for this frame
frm_erange	-11	Frame id specified was not defined with Create_Frame
bcr_erange	-12	Run_BC called with type > 255
frm_maxframe	-13	Exceeded maximum number of frames permitted (20)
frm_erangecnt	-14	Message entry greater than number of messages in the frame
eintr	-15	Attempt to set an undefined interrupt
etiming	-16	Attempt to change a message while module is accessing that message
estackempty	-17	Attempt to read command stack before any messages have been received
enomsg	-18	Get_Next_Message_BCM called when no new, unread messages have been received
enoskip	-19	Attempt to restore a message that was not skipped
enoasync	-20	The async frame contains fewer messages than the user wants to send asynchronously
etimeout	-21	Module is currently running; module was not in halted state

Error Code	Value	Definition
einvamp	-22	Attempt to set invalid amplitude
eboardnotfound	-23	Too many modules initialized
eboardnotalloc	-24	Attempt to switch to segment which was not allocated in Init_Card
Init_Card_noid	-25	Init could not find a device/module at given segment or device number
etimeoutreset	-26	Timed out waiting for reset
frm_nostack	-27	Start_Frame was not yet called to set up the message stack
eboardtoomany	-36	Too many modules initialized
func_invalid	-49	Function invalid for this module
noirqset	-53	No interrupt allocated.
ilbfailure	-57	Internal loopback test failed
elbfailure	-58	External loopback test failed
rterror	-59	Illegal RT number selected
ebadcommandword	-60	If the T/R bit in the Command word is set incorrectly for the given cmdtype parameter
einit	-61	The module was not initialized
enoalter	-62	This message is being transmitted; do not alter the data
eopenkernelpci	-64	Cannot open kernel device (with PCI and cPCI boards)
enobcmsg	-65	Attempted to read BC message when no new messages have been received
einbcmsg	-66	Error in specified BC message
ertbusy	-67	Current entry in the RT message stack is being written to. Try again
econcurrmonmodule	-81	PCI: Modules 1 or 3 must be selected VME/VXI: Modules 1 or 3 or 5 or 7 must be selected
eoverrun	-82	Cannot read monitored message, messages were overrun
eminorframe	-84	The message specified is of the type Minor frame
edbnotset	-85	Double buffering not set
ebadblknum	-86	Bad data block number assigned for double buffering
exreset	-87	Xilinx failed to reset
ercvfunc	-88	This function valid for receive RTid only
ebadttag	-89	Time tag not working

Error Code	Value	Definition
ettagexternal	-90	External Time tag clock is not 4 μsec.
enosrq	-91	SRQ was disabled by the user
For PCI carrier boards only		
eopenkernel	-1001	Cannot open kernel device; check **`ExcConfig`** settings
ekernelcantmap	-1002	Kernel driver cannot map memory
ereleventhandle	-1003	Error in kernel Release_Event_Handle
egetintcount	-1004	Error in kernel Get_Interrupt_Count
egetchintcount	-1005	Kernel error in Get_Channel_Interrupt_Count when attempting to retrieve interrupt count
egetintchannels	-1006	Kernel error in Get_Interrupt_Channels when attempting to retrieve interrupt count
ewriteiobyte	-1007	Error in kernel WriteIOByte
ereadiobyte	-1008	Error in kernel ReadIOByte
egeteventhand1	-1009	Error in kernel mGetEventHandle, first part
egeteventhand2	-1010	Error in kernel mGetEventHandle, second part
eopenscmant	-1011	Error in openSCManager in startkerneldriver
eopenservicet	-1012	Error in openservice in startkerneldriver
estartservice	-1013	Error in startservice in startkerneldriver
eopenscmanp	-1014	Error in openSCManager in stopkerneldriver
eopenservicep	-1015	Error in openservice in stopkerneldriver
econtrolservice	-1016	Error in controlservice in stopkerneldriver
eunmapmem	-1017	Error in kernel unmapmemory
egetirq	-1018	Error in Get_IRQ_Number
eallocresources	-1019	Error allocating resources. See **`ReadMe.pdf`** for details on resource allocation problems.
egetramsize	-1020	Error in kernel GetRAMSize
ekernelwriteattrib	-1021	Error in kernel WriteAttributeMemory
ekernelreadattrib	-1022	Error in kernel ReadAttributeMemoryr
ekernelfrontdesk	-1023	Kernel frontdesk error
ekernelOscheck	-1024	Kernel Oscheck error
ekernelfrontdeskload	-1025	Kernel frontdeskload error
ekerneliswin2000compatible	-1026	Kernel is win2000 compatible error
ekernelbankramsize	-1027	Kernel banksize error
ekernelgetcardtype	-1028	Error in kernel GetCardType
emodum	-1029	Invalid module number specified

Error Code	Value	Definition
regnotset	-1030	Module not configured. Reboot after **`ExcConfig`** is run and board is in slot
ekernelbankphysaddr	-1031	Kernel GetBankPhysAddr error
ekernelclosedevice	-1032	Kernel CloseKernelDevice error
ekerneldevicenotopen	-1034	Kernel error: device not open
ekernelinitmodule	-1035	Kernel initialization error
ekernelbadparam	-1036	Kernel error: bad input parameter
ekernelbadpointer	-1037	Kernel error: invalid pointer to output buffer
ekerneltimeout	-1038	A Wait_For_Interrupt function returned with timeout
ekernelnotwin2000	-1039	Operating System is not Windows 2000
erequestnotification	-1040	Request_Interrupt_Notification error
ekernelnot4000card	-1041	Designated board is not a EXC-4000PCI
enotimersirig	-1042	Timers and IrigB not supported on this version of EXC-4000 board
eclocksource	-1059	If an invalid clock source was specified
eparmglobalreset	-1062	Illegal parameter used for `globalreset_flag` in StartTimer
etimernotrunning	-1063	Timer not running when ResetWatchdogTimer was called; did nothing
etimerrunning	-1064	Timer already running when StartTimer was called; did nothing
eparmreload	-1065	Illegal parameter used for `reload_flag` in StartTimer
eparminterrupt	-1066	Illegal parameter used for `interrupt_flag` in StartTimer
ebaddevhandle	-1067	Invalid handle specified. Use value returned by Init_Timers
edevtoomany	-1068	Init_Timers called for too many boards
einvalidOS	-1069	If an invalid operating system is used

For VME/VXI boards only

Error Code	Value	Definitions
eviclosedev	-1050	Error in ViClose device
evicloserm	-1051	Error in ViClose Default RM
eopendefaultrm	-1052	Error in ViOpenDefault RM
eviopen	-1053	Error in ViOpen
evimapaddress	-1054	Error in viMapAddress
evicommand	-1055	Error in VISA command
einstallhandler	-1056	Error in VISA viInstallHandler
eenableevent	-1057	Error in VISA viEnableEvent
eunistallhandler	-1058	Error in VISA viUnistallHandler
edevnum	-1060	If a device number greater than 255 was used
einstr	-1061	If a device was not successfully initialized by Init_Module_Px

Function Index

A

Alter_Cmd_Px, 5-5
Alter_IMG_Px, 5-6
Alter_Message_Px, 5-7
Alter_MsgSendTime_Px, 5-8
Assign_Blk_Px, 4-7
Assign_DB_Datablk_Px, 3-4
Assign_RT_Data_Px, 3-5

B

BC_Check_Alter_Msgentry_Px, 5-9

C

Clear_Card_Px, 5-10
Clear_Frame_Px, 5-10
Clear_Msg_Blks_Px, 4-2
Clear_RT_Sync_Entry_Px, 3-5
Clear_Timetag_Sync_Px, 5-11
Command_Word_Px, 5-12
Create_1553_Message_Px, 5-14
Create_Frame_Px, 5-13

E

Enable_1553A_Support_Px, 2-2
Enable_Checksum_Error_Px, 5-16
Enable_Checksum_Px, 5-15
Enable_Lkup_Int_Px, 4-7
Enable_SRQ_Support_Px, 5-16
External_Loopback_Px, 2 - 3

G

Get_BC_Msgentry_Px, 5-17
Get_BC_Status_Px, 5-19
Get_Blknum_Px, 3-6
Get_Board_Status_Px, 2-4
Get_Card_Type_Px, 2-4
Get_Checksum_Blocks_Px, 3-6
Get_Counter_Px, 4-10
Get_Error_String_Px, 2-5
Get_Header_Exists_Px, 2-5
Get_Header_Value_Px, 2-6
Get_Id_Px, 2-6
Get_Interrupt_Count_Px, 2-20
Get_Last_Blknum_Px, 4-8
Get_Message_Px, 4-8
Get_Minor_Frame_Time_Px, 5-19
Get_Mode_Px, 2-7
Get_Mon_Status_Px, 4-3
Get_Msgentry_Status_Px, 5-20
Get_Next_Message_BCM_Px, 5-22
Get_Next_Message_Px, 4-10
Get_Next_Message_RTM_Px, 3-7
Get_Next_RT_Message_Px, 3-9
Get_Rev_Level_Px, 2-7
Get_RT_Message_Px, 3-11
Get_RT_Sync_Entry_Px, 3-12
Get_Time_Tag_Px, 2-8

I

Init_Module_Px for PCI, 2-8
Init_Module_Px for VME, 2-10
InitializeInterrupt_Px, 2-21
Insert_Msg_Err_Px, 5-24
Internal_Loopback_Px, 2-11

L

Load_Datablk_Px, 3-13

P

Print_Error_Px, 2-12

R

Re_Create_Message_Px, 5-25
Read_Datablk_Px, 3-14
Read_Message_Px, 5-26
Read_RT_Status_Px, 3-14
Read_RTid_Px, 3-15
Read_SRQ_Message_Px, 5-27
Read_Start_Reg_Px, 2-12
Release_Module_Px, 2-13
Reset_BC_Status_Px, 5-27
Reset_RT_Interrupt_Px, 3-15
Reset_Time_Tag_Px, 2-13
Restart_Px, 2-13
RT_Id_Px, 3-16
Run_BC_Px, 5-28
Run_MON_Px, 4-3
Run_RT_Px, 3-16

S

Select_Async_Frame_Px, 5-28
Send_Async_Frame_Px, 5-29
Send_Timetag_Sync_Px, 5-29
Set_1553Status_Px, 3-17
Set_BC_Resp_Px, 5-30
Set_Bit_Cnt_Px, 3-18, , 5-30
Set_Bit_Px, 3-17
Set_Broad_Ctl_Px, 4-4
Set_Broad_Interrupt_Px, 3-18
Set_Bus_Px, 5-31
Set_Checksum_Blocks_Px, 3-19
Set_Cnt_Trig_Px, 4-12
Set_Continue_Px, 5-31
Set_Error_Location_Px, 5-32
Set_Frame_Time_Px, 5-32
Set_Halt_Px, 5-33
Set_Header_Exists_Px, 2-14

Set_Header_Value_Px, 2-15
Set_Interrupt_On_Msg_Px, 5-33
Set_Interrupt_Px, 2-16
Set_Invalid_Data_Res_Px, 3-19
Set_Jump_Px, 5-34
Set_Minor_Frame_Time_Px, 5-34
Set_Mode_Addr_Px, 3-20, , 4-4
Set_Mode_Px, 2-17
Set_MON_Concurrent_Px, 4-5
Set_Mon_Response_Time_Px, 4-6
Set_Replay_Px, 5-35
Set_Restore_Px, 5-36
Set_Retry_Px, 5-37
Set_RT_Active_Bus_Px, 3-21
Set_RT_Active_Px, 3-20
Set_RT_Broadcast_Px, 3-21
Set_RT_Errors_Px, 3-22
Set_RT_Interrupt_Px, 3-23
Set_RT_Nonactive_Px, 3-23, , 5-38
Set_RT_Resp_Time_Px, 3-24, , 5-39
Set_RTid_Interrupt_Px, 3-24
Set_RTid_Status_Px, 3-25
Set_Skip_Px, 5-39
Set_Status_Px, 3-26
Set_Stop_On_Error_Px, 5-40
Set_Sync_Pattern_Px, 5-40
Set_Timetag_Res_Px, 2-18
Set_Trigger_Mode_Px, 4-13
Set_Trigger1_Px, 4-14
Set_Trigger2_Px, 4-15
Set_Var_Amp_Px, 3-26, , 5-41
Set_Vector_Px, 3-27
Set_Wd_Cnt_Err_Px, 3-27
Set_Word_Cnt_Px, 5-41
Set_Zero_Cross_Px, 5-42
Start_Frame_Px, 5-43
Stop_Px, 2-18

W

Wait_For_Interrupt_Px, 2-22
Wait_for_Multiple_Interrupts_Px, 2-24

The information contained in this document is believed to be accurate. However, no responsibility is assumed by Excalibur Systems, Inc. for its use and no license or rights are granted by implication or otherwise in connection therewith. Specifications are subject to change without notice.

September 2004, Rev A-3